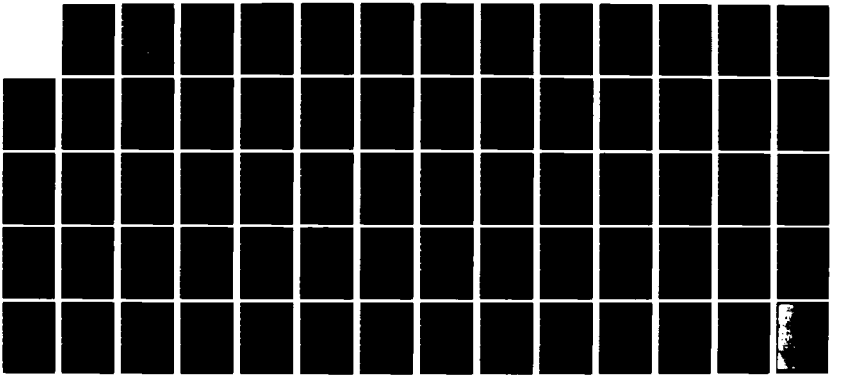
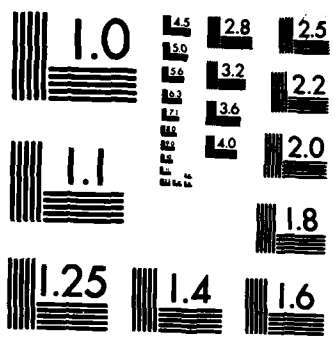
AD-A127 985 STATE-OF-THE-ART ASSESSMENT OF TESTING AND TESTABILITY OF CUSTOM LSI/VLSI..(U) AEROSPACE CORP EL SEGUNDO CA ENGINEERING GROUP M A BREUER ET AL. OCT 82 1/1

UNCLASSIFIED TR-0083(3902-04)-1-VOL-4 SD-TR-83-20-VOL-4 F/G 9/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A127985

State-of-the-Art Assessment of Testability of VLSI Circuits

Third Generation

DTIC FILE COPY

DTIC
ELECTE
MAY 1 0 1983
S D
E

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED

83 05 09-137

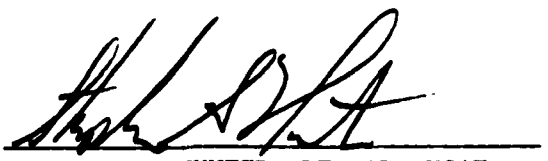
This final report was submitted by the Aerospace Corporation, El Segundo, CA 90245 under Contract No. F04701-82-C-0083 with the Space Division, Deputy for Logistics and Acquisitions, P.O. Box 92960, Worldway Postal Center, Los Angeles, CA 90009. It was reviewed and approved for The Aerospace Corporation by J. R. Coge, Electronics and Optics Division, Engineering Group. Al Carlan was the project engineer.

This report has been reviewed by the Office of Information and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication. Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

FOR THE COMMANDER

APPROVED



STEPHEN A. HUNTER, LT COL, USAF
Director, Speciality Engineering
and Test

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: SD-TR-83-20	2. GOVT ACCESSION NO.: AD-A127 985
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE *(and Subtitle)*: State-of-the-Art Assessment of Testing and Testability of Custom LSI/VLSI Circuits Vol IV: Test Generation	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER: TR-0083(3902-04)-1
7. AUTHOR(s): M.A. Breuer & Associates and A.J. Carlan Aerospace Technical Director	8. CONTRACT OR GRANT NUMBER(s): F04701-80-C-0081 F04701-81-C-0082 F04701-82-C-0083
9. PERFORMING ORGANIZATION NAME AND ADDRESS: M.A. Breuer & Associates 16857 Bosque Dr. Encino, CA 91436	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: Space Division Air Force Systems Command Los Angeles, Calif. 90245	12. REPORT DATE: October 1982
	13. NUMBER OF PAGES: 64
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: The Aerospace Corporation El Segundo, Calif. 90245	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Stuck-at-Fault	High Level Circuit Models	ARRAYS
D-Algorithm	TEST/80	Delay Testing
D-Algorithm Variants	Bit-Sliced Microprocessors	
SCIRTSS	Heuristic Testing	
PODEM	Signature Analysis	

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Two major approaches are considered for generating tests for digital systems: methods based on detailed circuit models of the unit under test (UUT) and methods based primarily on a functional description of the UUT. In addition to test generation of general digital systems, the testing requirements of microprocessors, semiconductor memories and PLA are examined. The D-algorithm and several variants are discussed as a basis for practical test generation procedures.

DD FORM (FACSIMILE) 1473

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

TABLE OF CONTENTS

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

SUMMARY OF FINDINGS

Test generation methods for digital systems may be divided into two major categories: those based on detailed logic-circuit models of the unit under test (UUT), and those based on high-level functional descriptions of the UUT. Most general-purpose test generation programs in current use implement some version of Roth's D-algorithm. These programs typically require a gate-level circuit description of the UUT and only yield tests for faults of the single line stuck-at-0/1 type. Many modifications to the D-algorithm have been developed to reduce its computation time. While the need for basing test generation methods on higher-level circuit models has been recognized, only a few limited attempts to implement such systems have been reported. A variety of different schemes have been developed for deriving tests from functional models of the UUT, particularly where the UUT is microprocessor-controlled. Heuristic methods employing self-test programs are widely used in such cases. Recently some promising work has been described that uses exact functional fault models for test generation in microprocessor-controlled systems. Considerable interest has also been shown in compact testing methods that involve pseudorandom test pattern generation and fault signature analysis. Finally, various testing procedures have been developed for specialized fault types found in LSI/VLSI designs, for example, high-density RAM faults and delay faults.

The problem of generating tests for single stuck-at-0/1 faults in combinational circuits is considered to be solved. Essentially complete test sets can be obtained for most combinational circuits, even those containing thousands of gates, using current implementations of the D-algorithm. Computational problems have been encountered with some kinds of code-checking circuits, but special methods to handle these circuits have been devised. Little attention has been given to test generation for other fault types, such as multiple stuck-line faults or short circuits. However, there is evidence that tests generated for the standard single stuck-line model provide good coverage for these other fault types. The D-algorithm has been successfully extended to deal with small and medium-sized synchronous sequential circuits containing up to about a hundred flip-flops. Poor results

are obtained for unstructured sequential circuits, such as those containing deeply buried flip-flops, due to the exponential growth of test computation time with the number of memory elements present. Asynchronous circuits pose even greater difficulties.

The D-algorithm has been successfully applied to sequential circuits in the LSI/VLSI range only when a highly-structured circuit design methodology like IBM's level-sensitive scan design (LSSD) has been followed. Besides being easily testable for the standard stuck-at faults, LSSD designs have proven particularly amenable to delay fault testing. The main drawback of LSSD is the slow rate at which test patterns must be applied; thus LSSD is not suitable for all applications. To handle large non-LSSD circuits, it appears necessary to develop computationally efficient test generation techniques that treat higher-level circuit components such as registers, multiplexers, ALUs, etc. as primitive. Although some research is being conducted into this problem, useful test generation programs are not yet generally available.

Very complex systems, such as those containing microprocessors, are usually tested in a heuristic fashion. Heuristic test generation methods attempt to test a device by systematically exercising all its major functions. While computationally efficient, the fault coverage of such approaches is uncertain, a consequence of the lack of an explicit fault model. Some work has been reported on fault models that are suitable for microprocessor-based systems. However, not all types of microprocessors or faults are included in this work. Considerable effort has been devoted to test generation for random-access memories, and a library of standard test algorithms has been compiled. Again fault coverage is unclear, particularly in the case of pattern-sensitive faults. While the use of compact testing methods such as Hewlett-Packard's signature analysis approach has been increasing, serious doubts have been raised about its fault coverage also.

Several aspects of test generation technology are likely to be pursued vigorously in the next five years. Methods of the D-algorithm type will probably be extended to accommodate more complex primitive elements. Easily testable design methodologies such as LSSD and built-in compact testing will see increased use, since they allow known test generation methods to be used for VLSI systems. However, it also is expected that emphasis will be placed

on the development of new design methods that lead to systems with better testability characteristics such as short testing time and 100% fault coverage. It is probable that there will be increased interest in test generation methods that can be incorporated into VLSI chips to make them self-testing, both to overcome IC pin limitations and to simplify field maintenance procedures. Finally it is expected that attention will be devoted to developing more complete detection methods for the newer fault types that are found in VLSI systems. Many of these faults, like stuck-at-open and parasitic flip-flop faults in CMOS circuits,are not covered at all by traditional fault models.

1. INTRODUCTION

This report surveys and assesses the methods currently used or proposed for generating tests for digital systems. Two major approaches are considered: methods based on detailed circuit models of the unit under test (UUT), and methods based primarily on a functional description of the UUT. The latter appear to be increasing in importance due to the proliferation of microprocessor-based systems where a complete circuit description is often unavailable to the test designer. In addition to test generation of general digital systems, the testing requirements of some important specific devices such as microprocessors, semiconductor memories and PLAs are examined. The influence of the fault models used on test generation is discussed, both for the standard stuck-at fault model and for such non-standard failure modes as pattern sensitivity and delay faults.

The test generation methods examined here are primarily intended for external testing. The use of built-in test equipment, information coding techniques or other design methods to enhance testability is not considered explicitly. However, the influence of such general design structures as level-sensitive scan design (LSSD) and bit-slicing on the test generation process is discussed where appropriate. The scope of the report is limited to the determination of the logical or functional correctness of the UUT. The generation of parametric tests is not considered.

Chapter 2 deals with test generation methods that require a detailed circuit description of the UUT. This description is usually a gate-level logic diagram. Test generation is typically directed at a set of faults defined on the logic circuit; the line stuck-at-0/1 model is the most common fault model used in this context. Section 2.1 reviews the D-algorithm which is the basis of many practical test generation procedures. Its use for testing both combinational and sequential circutis is discussed. Several variants of the D-algorithm are considered in Sec. 2.2 including the critical path method found in LASAR, a widely-used commercial test generation program. Also examined are SCIRTSS, a program written at the University of Arizona, and a recently-developed

program called PODEM which attempts to overcome the shortcomings of the D-algorithm when applied to error-correcting and -detecting circuitry. In Sec. 2.3 two test generation approaches that use register-level circuit descriptions are discussed: the LAMP system developed at Bell Laboratories, and a proposed system called TEST/80. Finally, in Sec. 2.4 the use of random test pattern generation is considered.

Chapter 3 is concerned with test generation that is based primarily on a functional description of the UUT or its major components, Sec. 3.1 considers methods that employ a very general functional fault model, including a testing scheme for bit-sliced systems, and a method of Akers based on binary decision diagrams. Heuristic test generation methods are the subject of Sec. 3.2. A typical ad hoc procedure for a microprocessor-based system is described, as well as a more formal approach due to Thatte and Abraham which employs a functional description called an S-graph. Section 3.3 surveys testing procedures that are grouped under the heading compact testing, and are characterized by the use of test response compression methods. Several representative compact testing methods are investigated, including transition counting, ones counting and Hewlett-Packard's signature analysis technique.

Chapter 4 is devoted to some special classes of circuits and their test generation problems. The failure modes and test methods employed for semiconductor memories and programmable logic arrays are reviewed. Some recent studies of pattern sensitive faults are discussed, as well as test generation methods for delay faults.

The report concludes with an extensive bibliography of recent literature on test generation.

2. CIRCUIT-BASED TEST GENERATION

In this chapter we examine and evaluate the available methods for test vector generation that employ an explicit circuit model which reflects both the structure and behavior of the digital system to be tested. Usually this model is designed at the logic gate level, with elements such as NAND, NOR, AND, OR, NOT, EXCLUSIVE-OR and flip-flops as the primitive building blocks from which the model is constructed. The LASAR test generation system, for example, requires a model that is composed entirely of NAND gates [Thomas 1971]. For LSI/VLSI circuits it is desirable to be able to use higher-level components such as registers, multiplexers, ALUs and the like as primitives during test generation. Recent work, for example, TEST/80 [Breuer & Friedman 1980], has been concerned with this problem. Also of recent concern is modeling those aspects of MOS ICs such as transmission gates, which do not correspond to standard logic gates [Wadsack 1978, El-Ziq 1979].

Circuit-based test generation methods may also be regarded as fault-based, since they generally employ specific fault models related to the circuit components that are recognized. The most common fault model is the stuck-at model in which any single logical connection in the circuit under consideration is allowed to be stuck-at-one (s-a-1) or stuck-at-zero (s-a-0). Other fault types such as multiple stuck-at faults, delay faults and short-circuit faults have received much less attention.

2.1 *THE D-ALGORITHM*

The D-algorithm developed by Roth at IBM is probably the most widely used test generation procedure [Roth 1966]; for a detailed description see [Breuer & Friedman 1976, Roth 1980]. The D-algorithm is a method for generating a test vector for a given fault. It is typically used with gate-level circuit models and stuck faults. The D-algorithm attempts to construct a sensitized path over which an error signal can propagate from the fault site to an observable primary output line. It systematically assigns values to lines associated with each potential sensitized path until a valid assignment is found, if one exists. Using a backtracking approach based on the circuit structure, the D-algorithm searches the space of possible test patterns for the given fault. The method in its most general form can always find a test

or, in the case of an inherently undetectable fault, prove that no test pattern for it exists.

COMBINATIONAL CIRCUITS

To illustrate the D-algorithm, consider the derivation of a test for the fault F = "output line of gate G_1 s-a-0" in the logic circuit of Fig. 2.1. The D-algorithm proceeds as follows.

(1) Assign values to the inputs of G_1 that produce a signal 1 at the output of G_1 when no fault is present; the values cd = 00 satisfy this requirement. The signal produced by G_1 is therefore 1 in the good circuit, but 0 when the fault F is present. (The special logic value D may be used to denote error-sensitive signals of this kind.)

(2) Select a path from G_1 to the primary output z, and attempt to assign input values to gates along this path so that an error signal can propagate from G_1 to z. In Fig. 2.1 the path shown by the heavy line is selected.

(3) Assign input values to all gates whose outputs are specified in Step (2) and check for logical consistency so that no line is assigned two conflicting values. Thus in Fig. 2.1, the value 1 must be assigned to all inputs of the NAND gates G_2 and G_4 that do not lie on the sensitized path. If the input combination cd = 00 is assigned to G_1 to produce the desired output value 1, then an inconsistency is encountered at G_3, which requires d = 1 to set its output signal to 1. The assignment cd = 01 satisfies all consistency requirements. The assignment of gate inputs continues until all primary input variables have been assigned values; these values constitute the desired test vector for F.

At all stages of the D-algorithm, including error propagation and consistency checking, line values are selected from many different possibilities. If a decision concerning some line leads to an inconsistency, it is necessary to backtrack to a previous decision point and select alternative values. In certain cases it may be necessary to sensitize several reconvergent paths for a particular fault. Such multiple-path sensitization cases are rare, however, and practical implementations of the D-algorithm are often restricted to single path sensitization in order to reduce computation time. In the

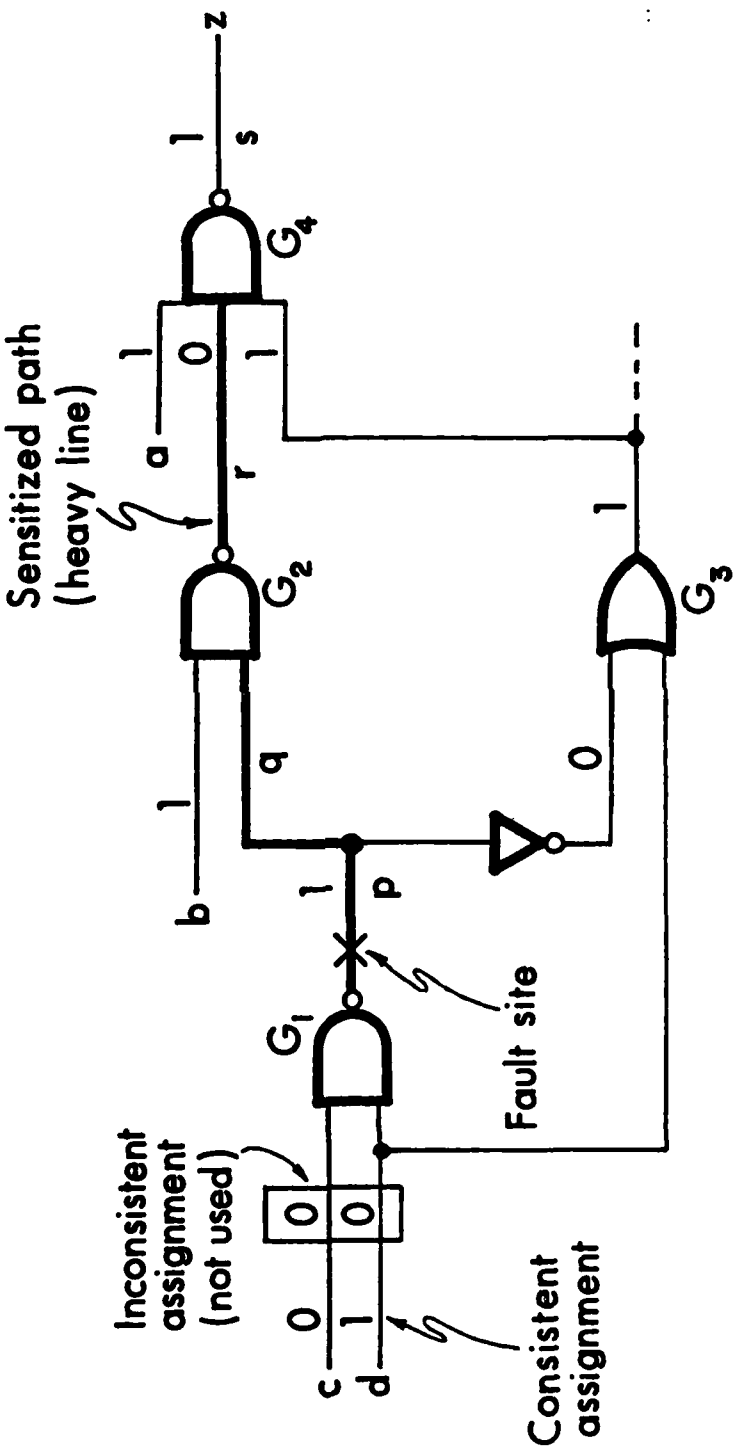


Fig. 2.1. Illustration of the D-algorithm.

worst case the computation time of the D-algorithm grows exponentially with circuit size; however worst-case behavior has only been observed for redundant circuits containing undetectable faults [Cha et al. 1978].

The D-algorithm is usually implemented in conjunction with a fault simulator program that can rapidly determine the faults detected by a given test vector. This simulator is used to compute all the hitherto undetected faults detected by each test vector produced by the D-algorithm. The basic D-algorithm incorporates five logic values, namely 0, 1, x (don't care) and the error-indicating values D and $\bar{D}$. The accuracy and flexibility of the D-algorithm, particularly when dealing with sequential circuits can be improved by using additional logic values. Several 9-valued implementations of the D-algorithm have been reported [Muth 1976, Cha et al. 1978]; a 16-valued version has also been investigated [Akers 1976, Breuer & Friedman 1976].

In conclusion, test generation for single stuck-at faults in combinational logic circuits containing several thousand gates can be effectively carried out using path sensitization type algorithms. Most large companies have successfully implemented this technique and use it for production test generation. This technique cannot efficiently handle multiple faults, but can process single faults other than stuck-at failures. The faults, however, must be representable within the given circuit model being employed.

SEQUENTIAL CIRCUITS

The basic D-algorithm discussed above applies to combinational logic circuits. It can be extended to synchronous sequential circuits by modeling them as iterative combinational arrays in the manner depicted in Fig. 2.2. The original sequential circuit S is partitioned into an acyclic combinational part C and a memory part M, the latter typically being a set of clocked flip-flops. The feedback from M to C is conceptually broken, and M is replaced by a (pseudo-) combinational version M*. The resulting pseudo-combinational circuit S* forms an accurate logical model of S during one time frame i. The combinational D-algorithm can be applied to S* to find a (partial) test vector T(i) for a given fault. By iterating S* k times as shown in Fig. 2.2b, a sequence of k input vectors T(0)T(1)...T(k-1), i.e., a sequential test, can be constructed via k applications of the D-algorithm.

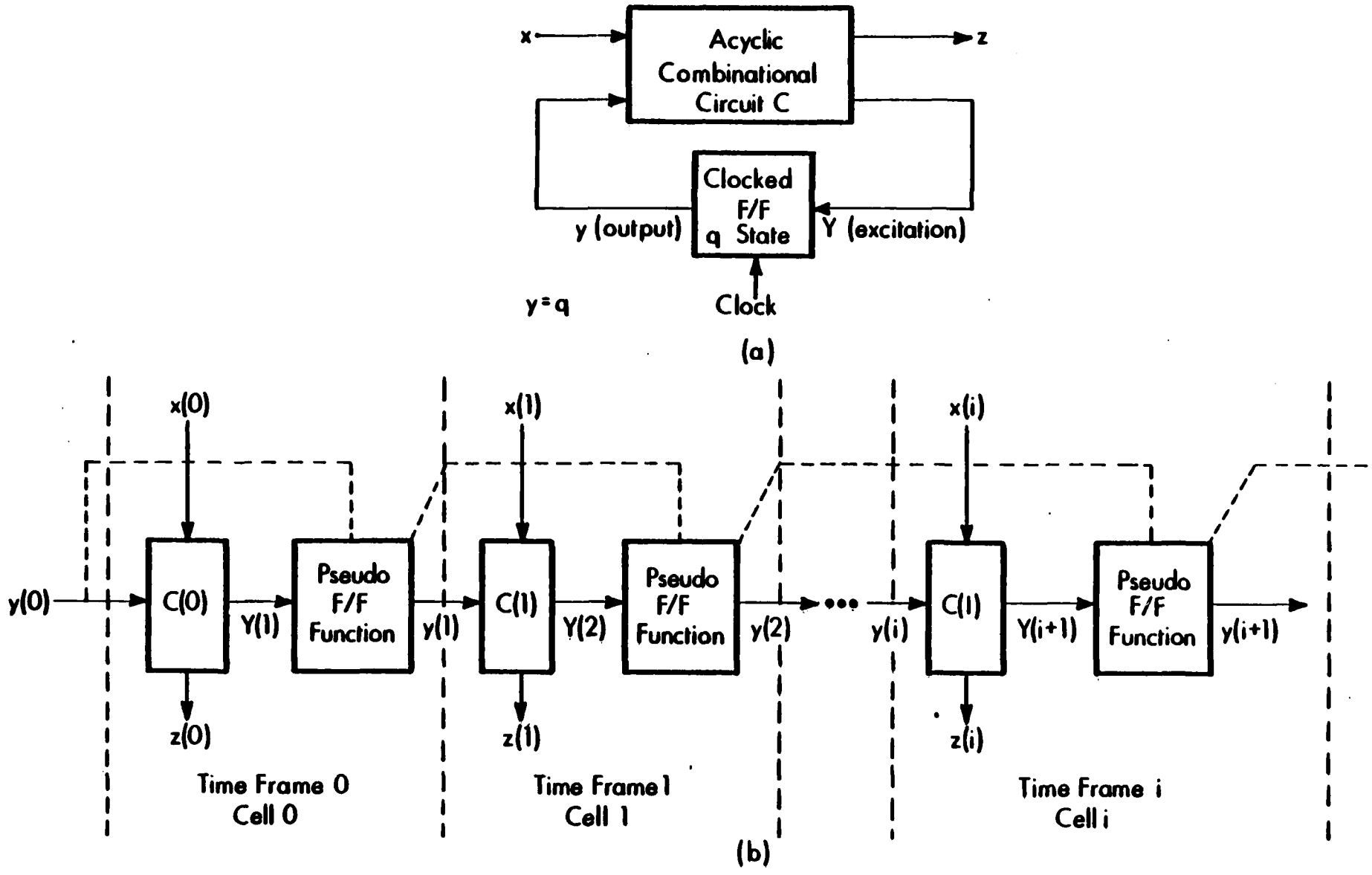


Fig. 2.2. Synchronous sequential circuit model for the D-algorithm.

There are several major problems associated with the use of the D-algorithm for sequential circuits. The length of a sequential test is not known a priori, and can grow exponentially with n, the number of flip-flops present. To keep computation time within reasonable limits, the searching processes associated with error propagation and consistency checking are often constrained by user-imposed limits. These limits can result in the failure to find tests for certain faults. Another difficulty arises from the fact that the initial state of the circuit under test is usually not known prior to testing. This situation can be handled by assigning the value U (for unknown) to uninitialized lines. The D-algorithm is therefore required to process up to 16 values for each line, representing all possible combinations a/b of signals in the good/faulty circuits, where $a,b \in \{0,1,x,U\}$. Even using 16 values, situations may exist where a test that is known to exist cannot be determined by the D-algorithm [Breuer & Friedman 1976]. For an example of a typical industrial automatic test generation program based on the D-algorithm, see [Yamada et al. 1977]. This program, developed at Nippon Electric, handles circuits with 1200 gates and 80 flip-flops, achieving an average fault coverage of about 95%. It employs a 10-value version of the D-algorithm and works in conjunction with a fault simulator and a random test pattern generator.

Asynchronous sequential circuits are especially difficult to process using the D-algorithm [Putzolu & Roth 1971]. Because of the possibility of unstable intermediate states in transitions between stable states, each time frame (see Fig. 2.2) may have to be expanded into several time frames to account for the unstable states. Furthermore, a test generated by the D-algorithm for an asynchronous circuit may be invalidated by the presence of a race or a hazard condition. The D-algorithm has been modified to allow the generation of hazard-free tests [Breuer & Harrison 1974, Vaughn 1976].

It is worth noting that the D-algorithm is particularly well-suited to test generation for circuits designed using LSSD (level sensitive scan design) and similar techniques [Williams & Angell 1973, Yamada et al. 1977, Eichelberger & Williams 1978]. In this design methodology, the flip-flops are linked to form a directly accessible shift register during testing. The circuit can be immediately initialized to any desired state, and the next state at the end

of any time frame can be immediately observed. Thus the testing problem reduces to one of finding test vectors for the combinational portion of the circuit; this is readily solved by the D-algorithm, even in the case of LSI circuits containing thousands of gates.

2.2 *VARIANTS OF THE D-ALGORITHM*

The D-algorithm is the basis of a large number of practical test generation programs which contain significant deviations from Roth's original version. Three representative examples are examined in this section: the critical path method which attempts to construct sensitive paths and only implicitly deals with faults, a program called SCIRTSS which uses high-level circuit models to speed up test generation for sequential circuits, and PODEM which employs branch-and-bound techniques and is aimed at the special test generation problems of circuits that process error-correcting and -detecting codes. Other interesting variants of the D-algorithm also exist, for example SOFTG (Simulator Oriented Fault Test Generator) [Snethen 1977].

CRITICAL PATH METHOD

The widely-used commercial test generation systems LASAR and D-LASAR employ a modified version of the D-algorithm called the critical path method [Thomas 1971, Breuer & Friedman 1976]. It is based on the fact that when a path P in a circuit is sensitized, easily identified faults associated with every line of P are detected. For example, in addition to the target fault F = line p s-a-0, the test shown in Fig. 2.1 detects the faults q s-a-0, r s-a-1 and s s-a-0, all of which lie on the indicated sensitized path. Intuitively speaking, many faults of this type can be detected simultaneously by maximizing the number and length of the sensitized paths produced by a given input pattern.

The critical path algorithm attempts to construct many long sensitized paths simultaneously. Path sensitization begins at a primary output, to which the value 0,1 is assigned. Proceeding backwards towards the primary inputs, values are assigned to gate inputs to produce a previously assigned output in such a way that one or more sensitized paths from the primary output are extended through the gate. For example, let 1 be assigned to the output of gate G_4 in Fig. 2.3. If pq = 10 is assigned to inputs of G_4 as shown, then a

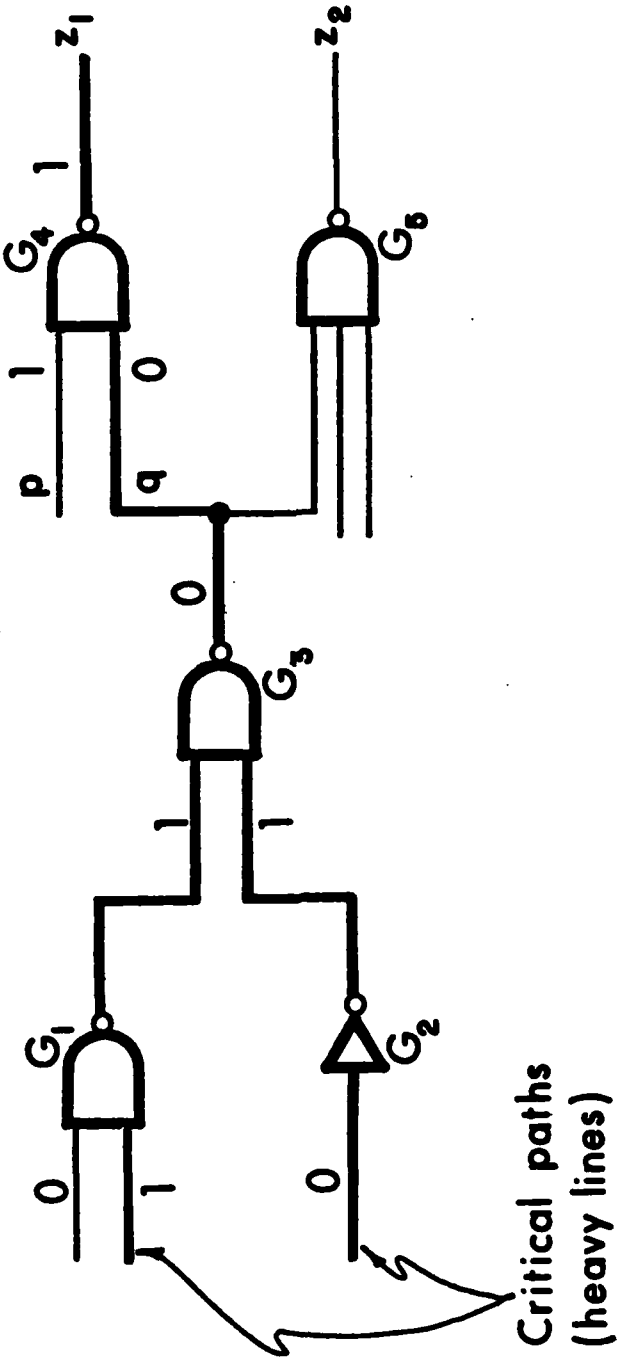


Fig. 2.3. Illustration of the critical path method.

sensitized or "critical" path is extended from z_1 to line q; the value 0 assigned to q is termed a critical value. Continuing backwards, we can assign the critical values 11 to the inputs of G_3, and so on. In general, if a line on a critical path has the value 0 (1) and is s-a-1 (s-a-0), then an error will propagate to a primary output. Hence this procedure implicitly deals with stuck-at faults. As in the D-algorithm, backtracking is used to ensure that all lines affected by the construction of the critical paths are assigned consistent values. The algorithm is applied repeatedly using all primary outputs of the circuit in turn, until critical 0's and 1's have been assigned to as many lines as possible. Clearly if a critical $d \in \{0,1\}$ is assigned to some line L by a test T, then T detects the fault L s-a-$\bar{d}$.

Since the critical path algorithm often generates only single sensitized paths from any given fault site, it may fail to find tests for certain faults, where the D-algorithm will succeed. It should be noted too, that the D-algorithm can be implemented in a way that also tends to produce long sensitized paths. This can be done, for example, by selecting the initial fault sites close to the primary inputs, or by assigning values during the consistency check phase to extend sensitized path length wherever possible.

Like the D-algorithm the critical path sensitization algorithm can be extended to sequential circuits by using an iterative array model. It too encounters the same problems at the D-algorithm in terms of circuit complexity, races and hazards. Though the D-LASAR program uses a NAND gate model of a circuit, the basic concepts behind the critical path sensitization approach are applicable to more general circuit models.

SCIRTSS

Hill and his colleagues have developed a test program called SCIRTSS (Sequential Circuit Test Search System) designed for moderately large (up to 60 flip-flops and several hundred gates) and unstructured sequential circuits [Hill & Huey 1977, Huey 1979]. The main contributions of SCIRTSS are the use of a register transfer language (AHPL) for input circuit description, and a fast heuristic search procedure to determine the state transition sequences required for fault propagation. A low-level (gate and flip-flop) model is also used, as well as the corresponding stuck-fault model. SCIRTSS has been implemented on a CDC 6400 at the University of Arizona.

Figure 2.4 outlines the organization of SCIRTSS. It is composed of four main sections.

(1) A version of the D-algorithm is applied to a pseudo-combinational gate-level version of the circuit under test with some selected fault F present. This identifies the primary input line values and input (control register) state S required to propagate an error signal from F to the primary output lines or the next state S′.

(2) Using a high-level circuit model, SCIRTSS executes a "sensitization search" program that attempts to find a state transition sequence leading from a previously determined state S_0 to the state S computed by the D-algorithm. S_0 is assumed to be unknown the first time this procedure is executed.

(3) A "propagation search" program is then executed to find an input sequence that eventually drives a fault signal from S′ to a primary output line. This completes the test generation process for F.

(4) A conventional parallel simulator is invoked to identify the previously undetected faults that are detected by the test generated for F. The list of undetected faults is updated, and a new undetected fault is selected for test generation.

The two search routines of SCIRTSS are similar, and are implemented using standard AI techniques. The space being searched is a tree-like representation of the state behavior of the circuit under test. The tree description is in high-level form based on AHPL, and is relatively compact. Weights are assigned to nodes of the tree according to the heuristic formula

$$H_n = G_n + \omega F_n$$

where G_n is the distance from the starting node to the current node n, and F_n is a user-defined value of weight ω. Typically large values of F_n are assigned to flip-flops near observable primary output lines, so that SCIRTSS favors error propagation towards the observable outputs.

SCIRTSS appears to have successfully achieved its goal of efficient test vector generation for sequential circuits containing several hundred gates. For example, SCIRTSS was able to find 99.4% of the faults in a 4-bit microprocessor slice containing 240 gates and 36 flip-flops [Hill & Huey 1977]. It

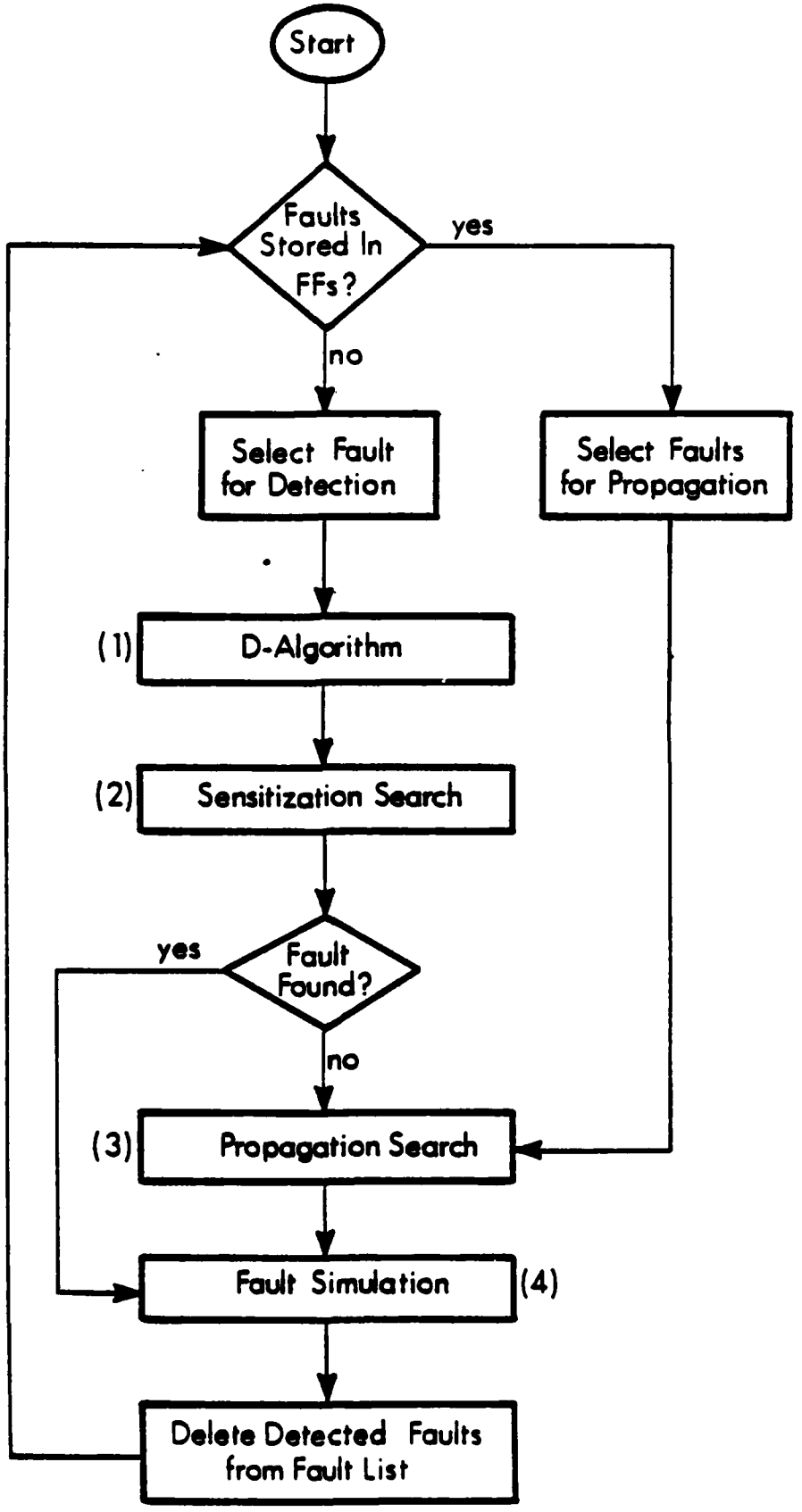


Fig. 2.4. Flowchart for SCIRTSS

has been estimated by its designers that by using a high-level circuit representation, SCIRTSS achieved a tenfold speed improvement over conventional gate-level test generation programs. Several limitations of the SCIRTSS technique should be noted. The search processes based on high-level language descriptions can be inaccurate; it may be difficult to provide good heuristics to guide the search. The search routines are also able to handle only one fault at a time. It would be desirable to process many faults simultaneously.

PODEM

The D-algorithm can be very inefficient when applied to circuits composed mainly of XOR (exclusive-or) gates: such circuits are commonly used in error-correcting and -detecting logic. Figure 2.5 illustrates the problems that can arise. A test vector is to be generated for the fault F = m s-a-0. Suppose that the D-algorithm applies 1 to m and sets $k = \ell = 1$ to create the indicated sensitized path to z. Since the functions appearing on k and ℓ are complementary, this assignment is inconsistent. However, the D-algorithm will not find consistent value for k and ℓ until it has backtracked to the inputs of the 4-input XOR or parity trees feeding k and ℓ, and tried up to half the 2^4 possible input combinations to these parity trees. In general, with n-input parity trees the D-algorithm may enumerate up to 2^{n-1} input signal combinations to find tests for certain faults.

To circumvent the foregoing problem, Goel of IBM has recently implemented a test generation algorithm called PODEM (Path Oriented Decision Making) [Goel 1980]. PODEM is basically a branch-and-bound algorithm for systematically searching the solution space of all possible input patterns for a test for each given fault. Experimental results reported by Goel indicate that PODEM can achieve 100% fault coverage for XOR-type combinational circuits at speeds substantially faster than the D-algorithm. Surprisingly, Goel claims that PODEM's performance is also better than the D-algorithm in the case of more general circuits; this has not been independently verified, however. It should also be noted that XOR circuits have properties that make them amenable to special-purpose test generation procedures that are far simpler than PODEM. For example, a test set of only four vectors can easily be constructed that detects all single faults in any simple XOR tree [Hayes 1971].

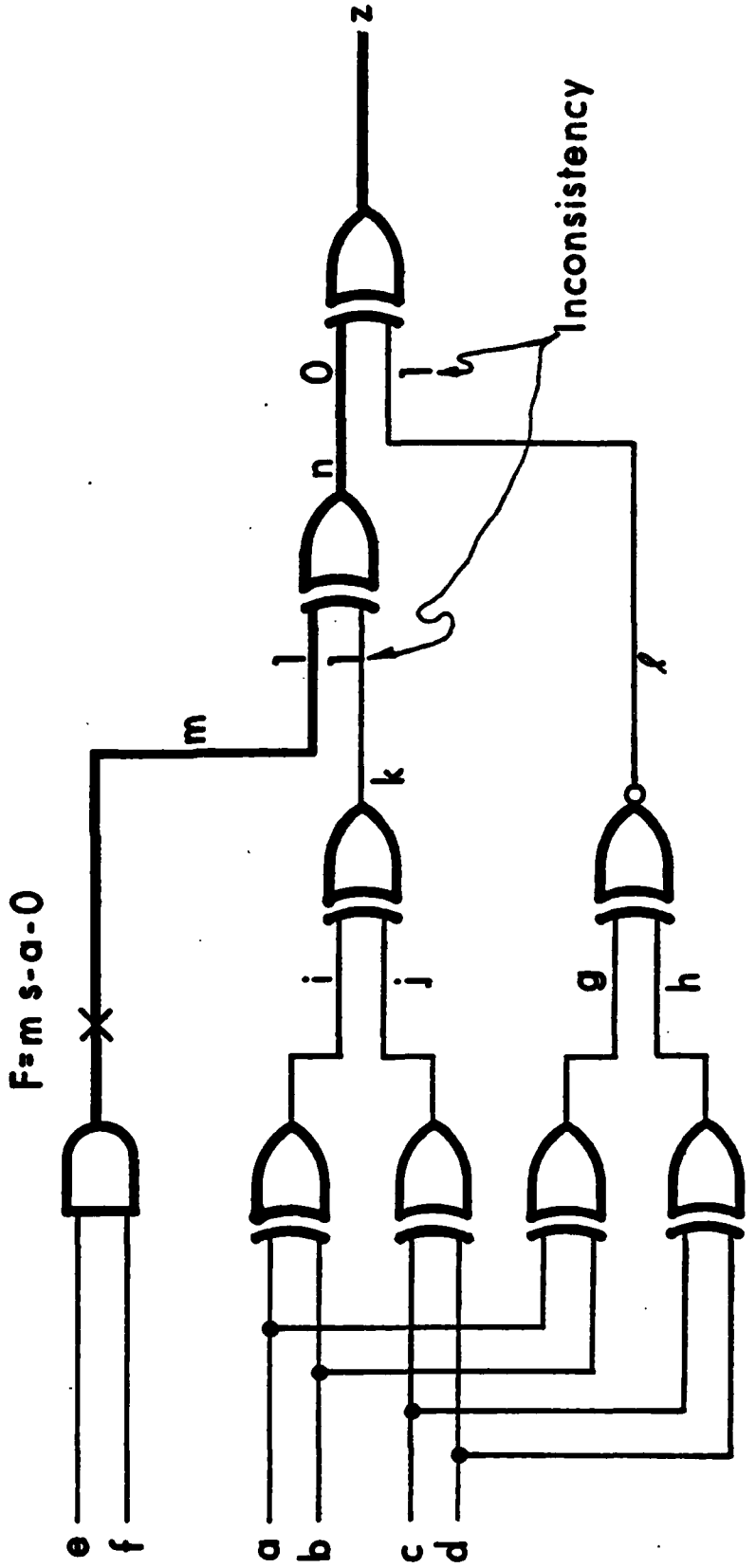


Fig. 2.5. Test generation for an error-detecting logic circuit.

2.3 *HIGH-LEVEL CIRCUIT MODELS*

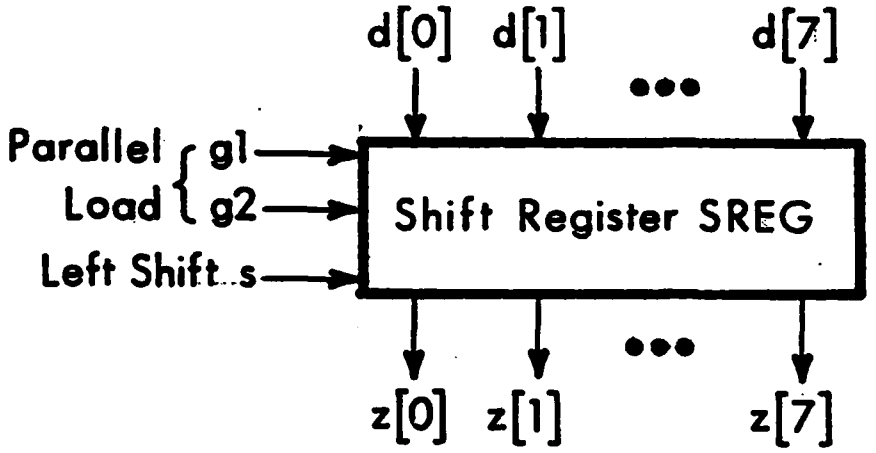
The test generation methods discussed in the preceding sections are primarily useful for gate-level circuits. Although complex circuit modules can, in principle, be accommodated by the D-algorithm, the bit-level behavior descriptions (D-cubes) required for such modules become very cumbersome. Recently increased attention has been devoted to test generation using circuit models containing complex components such as registers, ALUs and the like as primitive elements. This work is motivated by the need to reduce test generation time in complex LSI/VLSI systems, and by the fact that gate-level descriptions of the system being tested are often unavailable. Relatively few such test generation methods have been implemented. Two approaches are described here: the LAMP system implemented at Bell Laboratories, and a proposed method called TEST/80.

LAMP

The LAMP system developed at Bell Laboratories provides extensive facilities for simulating circuits containing both gate-level and register-level elements during test generation [Chappell et al. 1977]. Circuits are described using two special-purpose languages: FDL for register-level circuits, and LSL-LOCAL for gate-level circuits. Figure 2.6 shows an FDL description of an 8-bit shift register named SREG. Faults may be inserted into FDL descriptions by treating the fault name f as a variable, and inserting a functional description of the fault within an FDL compound statement of the form

```
IF f THEN AT {fault actions}
     ELSE AT {fault actions}
     ...
```

LAMP can process any fault defined in this way within a module, as well as the standard stuck-type faults. At most one fault is allowed to be present at any time. However, LAMP is capable of simulating a very large number (approximately 32,000) of single faults simultaneously. LAMP, which is available only within Bell Laboratories, has been successfully employed for design verification and test generation in relatively large digital systems where use of a pure gate-level model would have been prohibitively expensive. LAMP is primarily a fault simulator rather than an automatic test generation system.



(a)

```
REG: sreg;
SIZE: 8;
PREFIX: d,z;
INPUTS: g1, g2, s, d[0-7];
OUTPUTS: z[0-7];
HISTORY: 2;
DEF:
IF g1
THEN z[0-7] = sreg⟨0-7⟩
ELSE z[0-7] = s1;
  IF g2
  THEN sreg⟨0-7⟩ = d[0-7]
  ELSE
    IF ¬s(-1) & s
    THEN sreg⟨1-7⟩ = sreg⟨0-6⟩; sreg⟨0⟩ = 0
    FI
  FI
FI
FED
```

(b)

Fig. 2.6. (a) An 8-bit shift register and (b) its description in FDL.

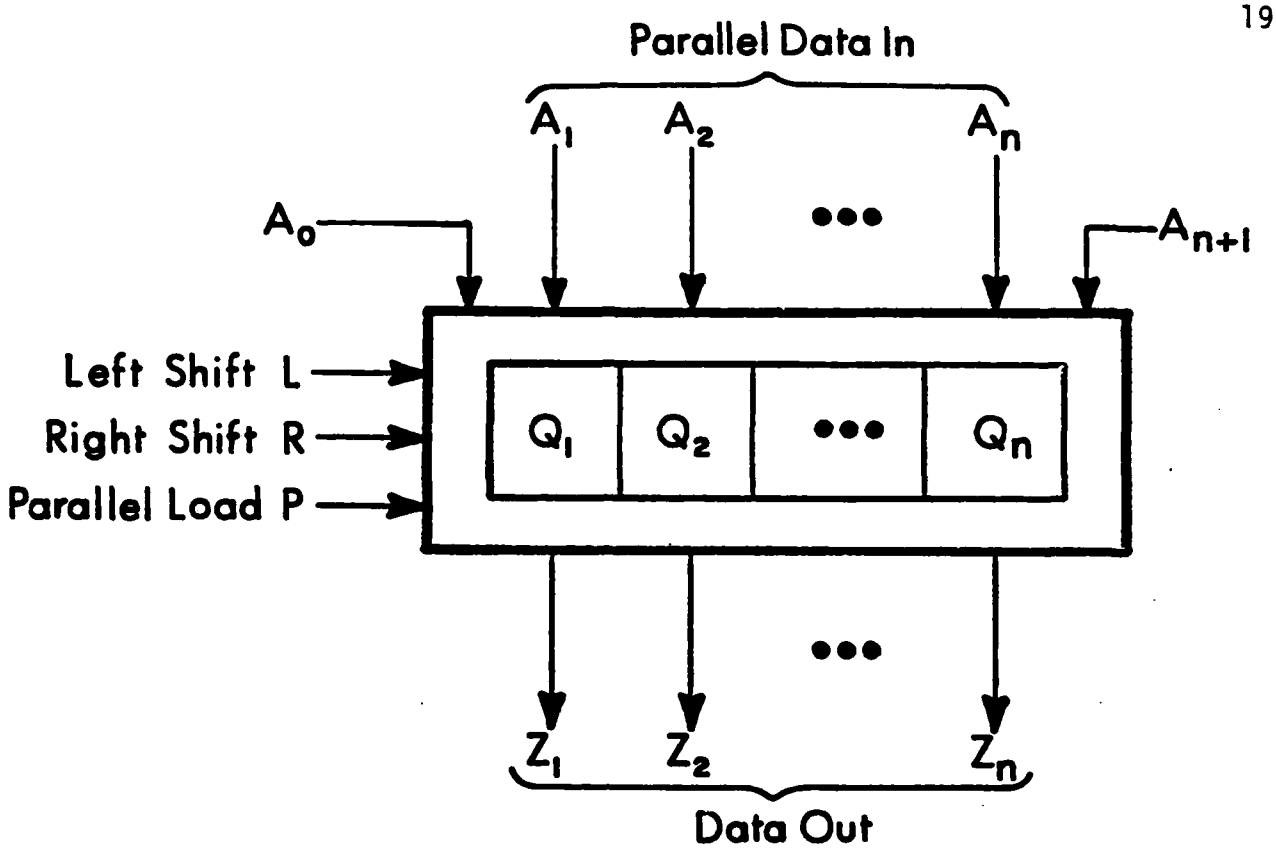
TEST/80

The recently proposed TEST/80 system [Breuer & Friedman 1979,1980] generates tests for circuits containing register-level components using a procedure based on the D-algorithm. Logic elements such as multiplexers, decoders, shift registers, counters and RAMs can be treated as primitive elements. TEST/80 is primarily concerned with reducing the computation time required for test generation. The system, which has not yet been implemented, uses high-level languages for both circuit and test description. Tests are represented by symbolic expressions that resemble regular expressions. This approach allows complex tests to be represented concisely as the solutions to implication, line justification and similar problems arising during the application of the D-algorithm to high-level circuit models. The main disadvantage of TEST/80 appears to be the large amount of effort required to develop models for the primitive components used.

To illustrate the use of FSL (Function Solution Language), the language used for test derivation in TEST/80, consider the left-right shift register of Fig. 2.7, where A, Q and Z denote the input, internal state, and output lines. The following type of line justification problem is typical of circuit-based testing methods: find an input (test) sequence that sets Z_1 to 1. In other words, justify $Z_1 = 1$. A simple solution to this problem is to load 1 into the shift register via the parallel data input A_1. This requires the parallel load control line P to be activated. This solution to the line justification problem can be expressed as follows in FSL:

$$\{A_1 \leftarrow 1, P\}.$$

Figure 2.7b shows a second FSL solution. In this case, a 1 is first loaded into the flip-flop Q_n of the shift register via the left-shift data input A_{n+1}; at the same time the left-shift control line L is activated. Then n-1 additional left-shift operations are carried out to move the 1 from Q_n to Q_1, thereby setting Z_1 to 1 as required. Each left-shift step can be separated by an arbitrary number of idle or hold (H) time periods, which are denoted by H* in the FSL solution.



(a)

Solution 1: $\{A_1 - 1, P\}$

Solution 2: $\{A_{n+1} - 1, L\}(H^*L)^{n-1}$

(b)

Fig. 2.7. (a) A left-right shift register (b) Two FSL solutions to the problem: justify $Z_1 = 1$.

Like the LAMP system, TEST/80 allows register-level primitives to be described by means of algorithms. These algorithms replace the tables of D-cubes typically used in gate-level implementations of the D-algorithm. They are used in carrying out three major steps of the D-algorithm, namely implication, D-drive (error signal propagation) and line justification.

TEST/80 also uses a new technique called cost analysis during circuit preprocessing prior to test generation. Cost analysis assigns numerical values to lines which are used to guide the various search procedures included in the test generation algorithm. Three cost values cA, $c\bar{A}$ and dA are associated with each line A that is an output of an element E. cA and $c\bar{A}$ denote the cost of setting A to 1 and 0 respectively, while dA denotes the cost of driving a D on line A to a primary output (D is Roth's error signal which is 1 in the fault free circuit, and 0 when a fault is present). In TEST/80, cA is defined by the equation

$$cA = \min(cfA + csA + cdA, K)$$

where cfA and cdA depend on the logic element E, and csA represents a "side effects" cost caused by fan-out from A. K is a large default value. $c\bar{A}$ is defined similarly. Algorithms have been developed for computing the costs associated with all the primitive elements of interest. Using these it is possible to compute the costs on all the lines in a circuit. In essence, cA, $c\bar{A}$ and dA are measures of observability and controllability associated with line A.

2.4 *RANDOM TEST GENERATION*

Another approach to circuit-based test generation is the use of a random pattern generator in conjunction with a fault simulator. A sequence of random patterns is generated for the circuit under test, and the fault simulator is used to evaluate the tests, e.g., to identify the stuck-at faults that they detect. Unlike methods based on the D-algorithm, tests are not generated for specific faults. The main advantage of random test generation (RTG) of this kind is the ease with which large numbers of potentially useful test patterns can be generated. Its disadvantages may be summarized as follows:

(1) Many faults require very specific, i.e., non-random, test pattern sequences to detect them, and hence may escape detection.

(2) RTG unlike test generation procedures based on the D-algorithm, cannot recognize undetectable faults, and may waste a great deal of time trying to generate tests for such faults.

(3) Random test sequences for a given set of faults are often much larger (a factor of 10 or more is common) than equivalent non-random tests for the same faults.

A general consequence of the foregoing problems is that the first test patterns in a random sequence tend to detect many faults, but the number of newly detected faults falls off rapidly. This is especially true for sequential circuits. This difficulty can be overcome by combining RTG with a method such as the D-algorithm [Breuer 1971, Agrawal & Agrawal 1972]. A recent example of this dual approach to test generation can be found in a program developed at Nippon Electric Company [Yamada et al. 1977]. It is claimed that tests for up to 70% of the faults of interest are generated by the NEC program during the RTG phase; the remainder are generated by the D-algorithm. The efficiency of RTG can also be improved by varying the probabilities with which 0's and 1's are applied to the input lines of the circuit [Schnurmann et al. 1975, Parker 1976b]. It is also usually desirable to be able to apply deterministic inputs to certain lines, e.g., the clock lines.

While RTG is useful for randomly-structured circuits such as are encountered in processors and control units, it is unsuitable for highly structured circuits like memories. In the case of combinational circuits, Agrawal has derived the following heuristic measure of the effectiveness of RTG [Agrawal 1978]:

$$M(99\%) = \ell n(0.01)/\ell n[1-q^{(n-1)L}] .$$

Here M(99%) is the number of random test patterns required for a 99% probability of generating a sensitized path of L levels, q is the probability of assigning 1 to each input line of the circuit, and n is the average fan-in of the circuit. M(99%) is easily computed for any combinational circuit. Agrawal suggests that when M(99%) is substantially less than 2^N, the total number of

possible input combinations, the circuit is a good candidate for RTG. For example in the case of the 74153, a standard multiplexer chip, $2^N = 4096$, while $M(99\%) \approx 200$, indicating that this circuit is suitable for random testing.

Interest in RTG has increased recently with the advent of compact testing. Compact testing methods such as transition counting and signature analysis allow very long random sequences to be used without generation of unmanageable quantities of response data, since the response data can be compressed into a compact signature. Furthermore testing can be done at the clock rate of the unit under test using very simple test equipment. Compact testing is considered in detail in Sec. 3.3.

3. FUNCTION-BASED TEST GENERATION

Gate-level circuit models of the type discussed in the preceding chapter are often unavailable for test generation purposes. This is particularly true of microprocessor-based systems, where often the only system descriptions available are a high-level block diagram and a listing of the microprocessor's instruction set. Even in cases where a more accurate gate-level description can be obtained, the number of gates present may be so great that too much computation is required for test generation at the gate level. This chapter examines the methods used for test generation based on high-level (typically at the register-transfer level) functional descriptions of the unit under test. While some work has been done using exact system and fault models, most test generation methods in this class are heuristic or ad hoc.

3.1 FUNCTIONAL FAULTS

Concern about the inadequacy of the conventional stuck-at-0/1 fault model in the context of MOS/VLSI [Wadsack 1978, Galiay 1980] has revived interest in the use of more general fault models. The most general such model of practical value has been termed the *functional fault model* [Sridhar & Hayes 1979] and may be defined as follows. Let M be a system or component module to be tested. Let M have s internal states and implement the function z. A permanent fault F is a functional fault of M if F changes M to M^F, where M^F realizes z^F and contains $s^F \leq s$ states. The fault is said to be detectable via an external test if $z^F \neq z$. This fault model includes all classical fault types such as stuck faults, short-circuit faults and pattern-sensitive faults, as well as many faults that have received little attention.

If M is a combinational circuit, i.e., if $s = 1$, then M^F must also be combinational. M is therefore tested for all functional faults of the above type by applying all 2^n possible input combinations to M, where n is the number of primary input lines present. This is, in effect, exhaustive testing which verifies the truth table of the circuit under test. Its complexity is clearly determined by the exponential growth rate of the 2^n input combinations, and appears to be limited to cases where $n \leq 32$ or so. This testing method is used by TRW for functional testing of its multiplier chips, which are implemented

as VLSI combinational arrays. The processing of the large amounts of test response data produced by this approach can be simplified by various data compression methods [Tsidon et al. 1978]; this is discussed later in the context of compact testing.

To detect all possible functional faults in M when M is a sequential circuit, i.e., $s > 1$, requires the use of a checking sequence [Friedman & Menon 1971]. A checking sequence must effectively verify the state table of M. It is only feasible to generate checking sequences of reasonable length if s is very small, or if M has an unusually simple structure.

BIT-SLICED MICROPROCESSORS

This approach has recently been applied with success to test generation for bit-sliced microprocessors [Sridhar & Hayes 1979, 1981]. A register-level description of the microprocessor to be tested is required. The circuit modules, such as registers, multiplexers and ALU are assumed to be subject to the foregoing functional faults; it is also assumed that at most one module is faulty at any time. Because of the short word size characteristic of bit-slices, the individual modules are sufficiently simple that it is feasible to use the checking sequence approach to derive a complete set of tests for all functional faults.

To illustrate this testing philosophy, consider the general-purpose microprocessor slice C appearing in Fig. 3.1 [Sridhar & Hayes 1981]. This is based on the widely-used AMD 2901 4-bit slice [Mick & Brick 1980]. C differs from the 2901 in having a 1-bit word size. This is less restrictive than it appears, since the tests for a 1-bit slice can readily be extended to larger slices or arrays of slices. Like the Motorola 10800 microprocessor slice, C has only two working registers. The ALU is a combinational module capable of twos complement addition and subtraction, and the standard logical operations. Circuits for carry lookahead arithmetic are not included; only ripple carry propagation between adjacent cells is used.

Test generation for C proceeds as follows. First a functional test T_i is derived independently for each module M_i in C. When M_i is an n-input combinational circuit T_i is simply the set of all 2^n n-bit input words. The ALU module M_F, for example, requires $2^6 = 64$ test patterns. Checking sequences of length 16 can be derived for the two 1-bit register modules M_A and M_T. Each module test T_i is then extended to a test T_i^* which, when applied to the primary

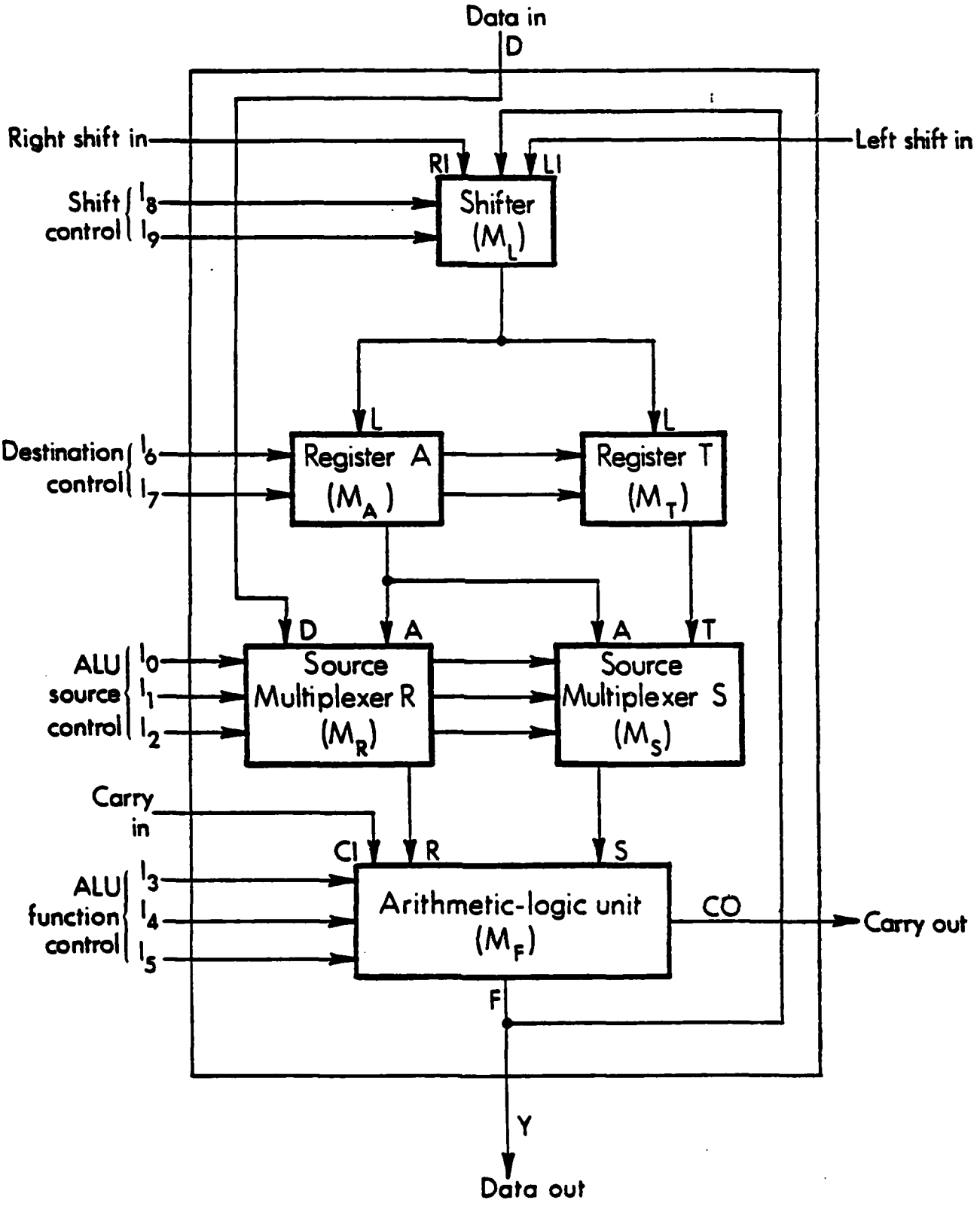


Fig. 3.1. A 1-bit microprocessor slice C.

inputs of C, causes T_i to be applied to M_i and M_i's responses to be propagated to observable outputs of C. An ad hoc path sensitization procedure is used to construct the T_i^*'s. This is a relatively simple task since C is a very simple circuit, i.e., it contains very few modules, at the complexity level being used. The various T_i^*'s can be combined to yield a test sequence T_C of length 114 which detects all single-module functional faults in C.

C has the useful property called C-testability [Friedman 1973] which means that an array of type-C cells of arbitrary length can be tested with a constant number of test patterns, in fact, the same number as is required for C itself. Sridhar and Hayes have extended C to a k-bit slice $C^{k,n}$ containing an $n \times k$ scratchpad RAM; $C^{k,n}$ requires, $64+2^{2k+4}+16n^2$ test patterns for complete functional fault coverage, assuming that the RAM is treated as a set of independent 1-bit registers. Thus $C^{4,16}$, which is very similar to the 2901, requires a test of length 8,256. This compares favorably with the 12,350 test patterns used by AMD to test this device [McCaskill 1976]; the AMD tests are completely heuristic, however, and their fault coverage is unclear.

While the foregoing testing philosophy is quite general, it depends on use of relatively simple functional modules such as appear in C. The Sridhar-Hayes approach exploits the regular interconnection structure found only in bit-sliced designs. It should also be noted that the cell C omits carry-lookahead, a speed-up feature of all commercial bit-sliced microprocessors. While the inclusion of carry-lookahead would not significantly increase the difficulty of testing C itself, the simple procedures for extending the tests for C to arrays of cells would no longer apply. It should also be noted that fast techniques exist for implementing ripple-carry arithmetic of the kind used in C, which reduce the speed advantages of carry-lookahead in the context of VLSI [Mead and Conway 1980].

BINARY DECISION DIAGRAMS

Another approach to test generation for functional faults has been proposed by Akers [Akers 1976, 1978] which makes use of binary decision diagrams (BDDs) to describe the functional behavior of a device being tested. Figure 3.2a shows a BDD for the simple combinational function

$$f = a + \bar{b}c .$$

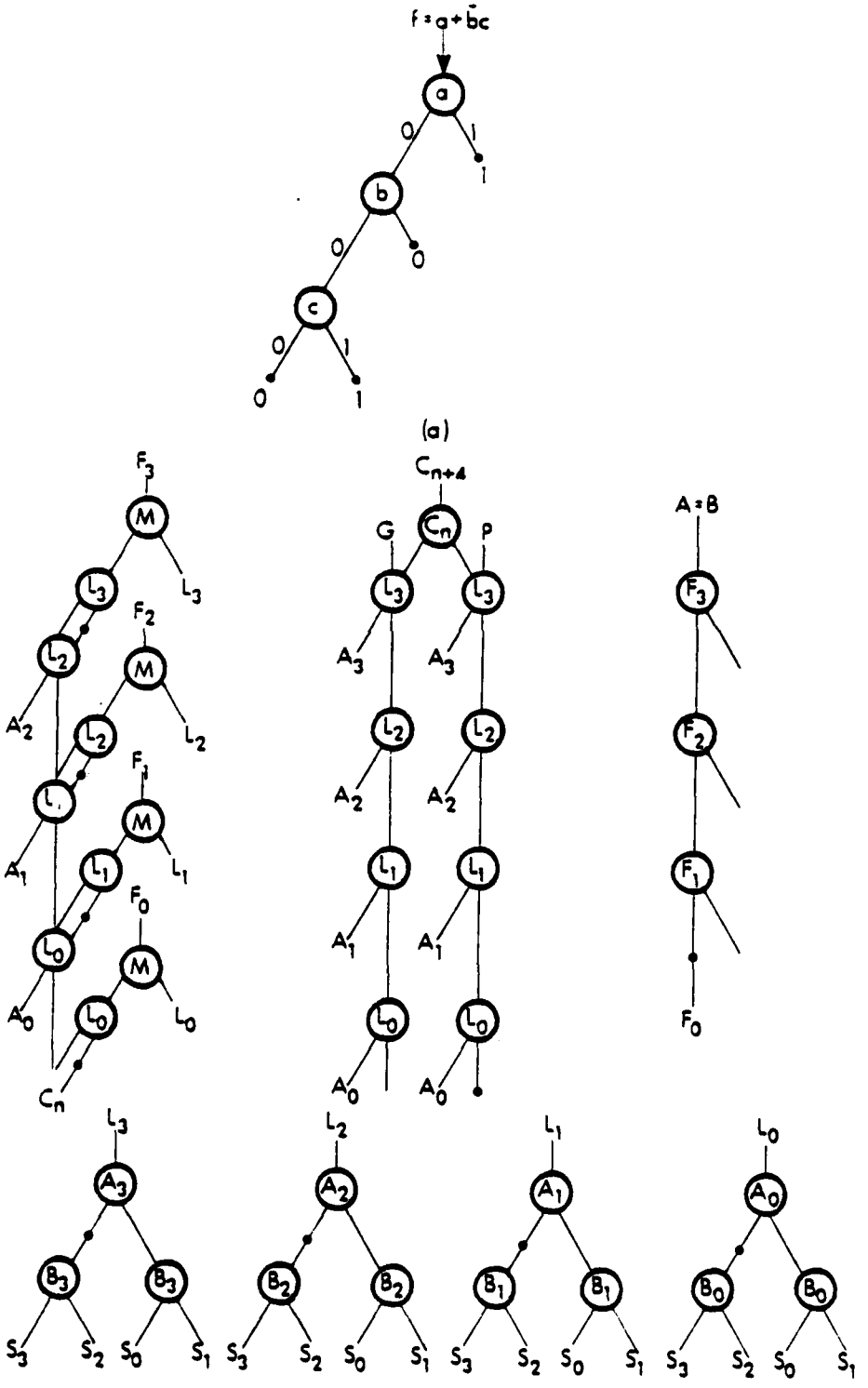


Fig. 3.2. (a) Binary design diagram (BDD) for the function $f = a+\bar{b}c$. (b) BDD for a 4-bit microprocessor slice.

The value of f for any combination of a, b, c can be found by selecting an appropriate path from the top (root) node of the BDD to a terminal node. Each non-terminal node represents an input variable; a branch to the left (right) at node n, corresponds to assigning a value of 0 (1) to n. The label on the terminal node reached is the output value of f corresponding to the path traced through the BDD. The main advantage of BDDs are their compactness. Figure 3.2b shows a 35-node BDD for a 4-bit microprocessor slice based on the AMD 2901 mentioned earlier [Mick & Brick 1980]. Akers asserts that for most devices of interest, the number of nodes in the BDD grows linearly with the number of device inputs.

A BDD may be used for functional test generation in the following way. All possible paths from an output f_i of the BDD to terminal nodes are computed. THe corresponding set of input patterns are collected to form a (partial checking) experiment [Akers 1978]. Node variables not included in a particular path are marked "don't care," while variables both of whose exit branches terminate at the corresponding node are marked as "sensitive." The experiments may be used to generate functional tests by varying their sensitive input variables in all possible ways, which effectively creates sensitized paths from these variables to the outputs of the circuit under test. This allows a portion of the circuit to be exercised in a systematic manner. Of course, a complete checking sequence must be applied if all functional faults are to be detected. The applicability of the BDD approach to high sequential circuits is questionable.

3.2 HEURISTIC TEST GENERATION

The usefulness of the foregoing testing approaches diminishes as the complexity of the unit under test increases. Systems with substantial numbers of LSI/VLSI components are often tested by heuristic means that merely attempt to exercise the major subsystems. Since many current systems contain microprocessors, there is considerable interest in testing methods that employ the microprocessor as the primary source of test patterns. This reflects the fact that the functions of the system are often described in terms of instructions or programs executed by the microprocessor. Thus it is possible to design a system test in the form of a suitably instrumented set of test programs executed by its internal microprocessor [Hayes & McCluskey 1980].

MICROPROCESSOR-BASED SYSTEMS

Heuristic testing of a microprocessor-based system typically proceeds as follows. First the microprocessor, which is the system's CPU, is tested. A typical CPU test [Chiang & McCaskill 1976] begins by executing instructions that cause the program counter PC to increment through all its states, a mode of operation called free-running. The PC state can be observed directly on the system address bus. Other CPU registers and buses are then tested by writing suitable test patterns into them and reading out the results via PC. If the microprocessor word size is relatively small, e.g., eight bits or less, then all possible data patterns can be applied to each register and bus. Note however that it is not feasible to apply every possible sequence of these patterns, since the number of possible sequences is too great. The CPU registers can be tested in the above manner by means of a small number of data transfer instructions. Next the ALU is tested by exercising all arithmetic, logical and conditional branch instructions. To complete the CPU testing, any previously unused CPU instructions, circuit components and IO lines must be exercised.

After testing the CPU, the system's main memory can be tested. ROMs are usually tested with the PC in free-running mode. The free-running PC places addresses on the system address bus causing the ROM contents to be placed on the system data bus where they can be observed and checked. Signature analysis is particularly useful here since a fixed set of fault signatures can be determined for the ROM. RAMs are usually tested by executing standard algorithms such as CHECKERBOARD and GALPAT which check for various types of functional failures [Breuer & Friedman 1976] (see also Sec. 4.3). Finally, the system's IO ports must be tested. This can be accomplished by a technique called loopback, whereby the system's output ports are temporarily connected to its input ports. This makes the IO portion of the system resemble a RAM, and RAM test patterns may be used to test it.

The main disadvantage of the foregoing testing procedure is the difficulty of measuring its effectiveness. There is no explicit fault model, hence no measure of fault coverage can be given a priori.

S-GRAPH MODEL

In an attempt to provide a more rigorous procedure for test generation via a system's instruction set, a microprocessor model called an S-graph was

developed recently [Thatte 1979, Thatte & Abraham 1980]. An S-graph describes the behavior of a microprocessor using only information that is provided in a user's manual. The nodes of the S-graph correspond to the registers $\{R_i\}$ of the microprocessor, including scratchpad registers, address registers, flag registers, etc. Two special nodes called IN and OUT denote the input and output ports connecting the microprocessor to main memory. If an instruction I_j causes data to flow from R_1 to R_2, then a directed edge (arrow) labeled I_j is drawn from R_1 to R_2. Since most instructions involve a sequence of register transfers, a superscript K is added to the edge label I_j to indicate that the corresponding register transfer occurs in the K^{th} step in the execution of I_j. Figure 3.3a shows a hypothetical microprocessor studied by Thatte and Abraham which has 21 instructions; Fig. 3.3b shows the corresponding S-graph.

Four general types of functional faults are defined in terms of S-graphs:

(1) Register decoding faults

(2) Instruction decoding faults

(3) Data storage faults

(4) Data transfer faults.

All these faults affect some general function associated with instruction execution, and they are largely independent of how the function is implemented. In most cases physical fault mechanisms can readily be found that give rise to the functional failures being modeled.

Register decoder faults occur when an operation that accesses (reads from or writes into) a register R_i, fails to access R_i, or accesses one or more incorrect registers R_j. When several registers are read simultaneously due to a register decoding fault, the accessed data is assumed to be either the AND or OR function of the various register contents. Register decoding faults correspond to stuck-at faults affecting multiplexers and demultiplexers in the data transfer paths between registers. The class of instruction decoding faults covers malfunctions of the instruction opcode decoding mechanism. An instruction I_j may be incorrectly decoded in three ways as specified by the following faulty behavior:

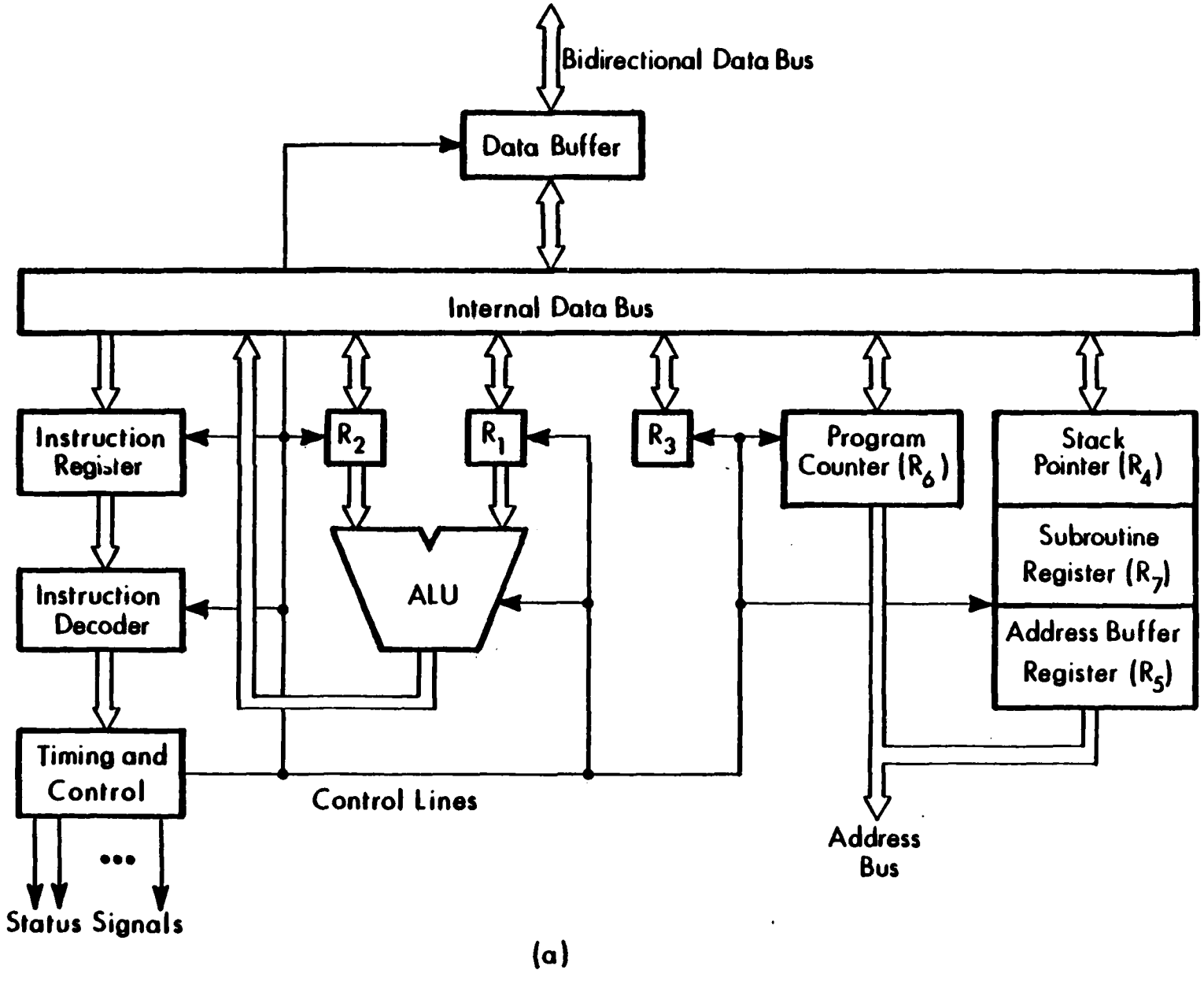


Fig. 3.3. (a) A hypothetical microprocessor. (b) The corresponding S-graph.

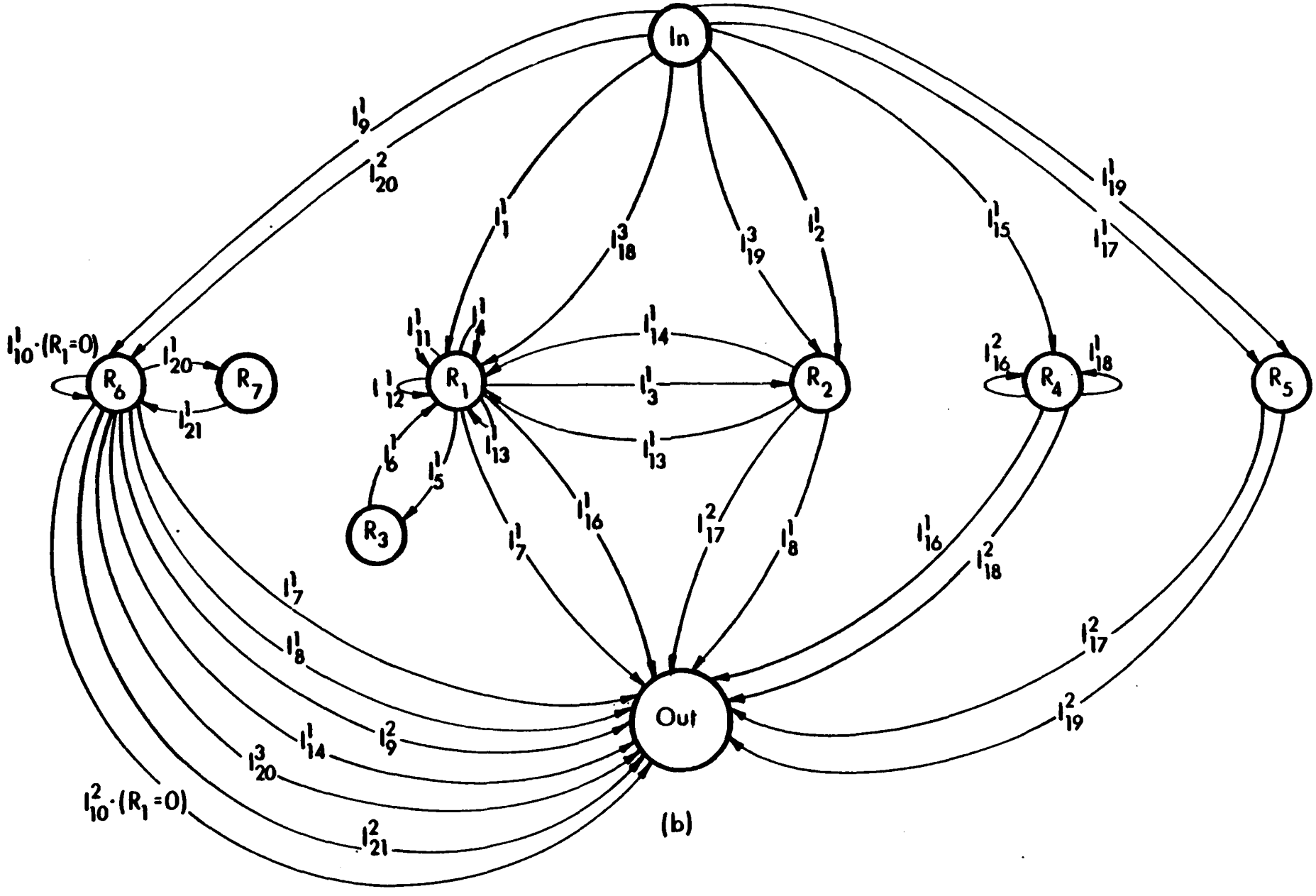


Fig. 3.3. continued

(1) Some instruction $I_k \neq I_j$ is executed instead of I_j.

(2) No instruction is executed.

(3) In addition to I_j, some other instruction I_k is executed.

Faults of this type correspond to physical failures in the circuitry that issues the instruction activation signals. Data storage faults refer to failures of the stuck-at-0/1 type in the registers themselves. Data transfer faults are associated with data transfer paths (buses). Two cases are permitted. One or more lines in the path may be stuck at 0 or 1. Two or more pairs of lines may be coupled so that they always carry the same logical value; this value is defined as the AND or OR function of the signals on the two coupled lines.

No specific fault model is proposed for "data manipulation faults" affecting processing circuits such as the ALU, address computation circuits, interrupt control, and the like. It is assumed that adequate functional tests for these faults are derived by other means using other fault models. These tests are added to the tests obtained from the model proposed. Finally, it is assumed that faults of only one of the above four types are present during testing.

Thatte and Abraham have constructed detailed test generation procedures for their four fault categories. In all cases the tests take the form of a sequence of instructions, i.e., a test program, to be executed by the target microprocessor. The tests are derived directly from the microprocessor's S-graph. It is assumed that testing is supervised by external test equipment which has direct access to the system bus or, equivalently, to the IN and OUT nodes of the S-graph. A typical test step involves selecting instructions that cause a data test pattern to enter the IN node and pass through various registers generating response data which is transferred to the OUT node. The test steps are ordered in such a way that the knowledge gained from the successful execution of early test steps can be used in later steps.

Thatte and Abraham have attempted to evaluate the feasibility of their approach by using it to generate test sequences for a microprocessor from Hewlett-Packard whose complete specifications have not been published. It contains 16 8-bit general-purpose registers and conventional special-purpose registers such as an 8-bit accumulator, an 11-bit program counter and an 11-bit stack pointer.

It has approximately 190 instructions. This microprocessor was chosen because the authors were able to obtain a detailed gate-level logic simulation model for it from Hewlett-Packard.

Applying their test generation procedures to this processor, Thatte and Abraham obtained (by hand) test programs containing about 9K instructions. Of these, about 8K were required to test for instruction decoding faults. The number of stuck-at faults detected by these tests was determined by using the TESTAID III fault simulator and the gate-level simulation model. About 2200 faults were simulated (which faults they were is not stated). It is reported that essentially all the simulated faults were detected. The 1K instructions constructed for faults other than instruction decoding faults detected about 90 percent of the simulated faults.

The approach presented is quite general and is applicable, in principle, to most microprocessors. The coverage of register decoding, data storage and data transfer faults appears to be very good. Furthermore, the necessary test programs are relatively easy to generate, and this process can probably be automated. The coverage of the instruction decoding fault tests is unclear. This part of the fault model appears to be unrealistic in several respects. It is not directly applicable to microprocessors such as the Motorola 68000 which use microprogrammable control. THe most serious apparent deficiency of this approach is the lack of a fault model for data manipulation faults. It is doubtful that such faults can be adequately treated using an approach based on the S-graph system model. Thus an alternate procedure to cover these faults must be used to augment this basic approach. In general, it is not easy to relate faults based on S-graphs to standard stuck-at faults; some relatively simple stuck-at faults are not covered.

3.3 COMPACT TESTING

The term compact testing [Parker 1976a] refers to a class of testing methods in which test response data is compressed (compacted) into a relatively short fault signature that can easily be compared to a fault-free reference signature. The terms signature testing and syndrome testing are also used for compact testing, but usually have narrower interpretations. A type of compact testing found in some commercial test equipment is transition counting, in which a signature is the total number of times a particular binary signal

changes state (from 0 to 1 or from 1 to 0) while a fixed sequence of input patterns are being applied [Lyons 1974]. A more sophisticated approach based on algebraic coding techniques and called signature analysis has been successfully promoted by Hewlett-Packard [Gordon & Nadig 1977]. Compact testing is used primarily with heuristically generated test patterns. These include pseudo-random patterns, and exercising patterns of the sort discussed in the preceding sections. Several analyses of compact testing have been published and are reviewed in the sequel. An important general characteristic of compact testing methods is that they require very simple test equipment. Hence they are very well suited to field testing and incorporation into built-in test equipment. Figure 3.4 shows the equipment typically used for compact testing.

Suppose that T is an input test sequence that produces a response sequence R at some test point P_1 of the unit under test. R is processed in real time by a suitable compaction technique to yield a signature f(R). f(R) is then compared to a precomputed signature $f(R_0)$. If $f(R) \neq f(R_0)$, then a fault is present in the circuits feeding P_1. Standard signal tracing techniques may be used to isolate the fault by obtaining the signatures at a sequence of test points $P_1, P_2, \ldots$. In general it is desirable to construct T so that $f(R) \neq f(R_0)$ for all faults of interest. Deterministic testing methods aim at obtaining T with 100% fault coverage. Note that $f(R) \neq f(R_0)$ implies $R \neq R_0$, so a compact test T must also be a test in the conventional sense. The problem of finding a compact test for a given set of faults can therefore be expected to be more difficult than the corresponding problem for non-compact testing.

TRANSITION AND ONES COUNTING

Transition count (TC) testing has been subjected to formal analysis, mainly for combinational circuits [Hayes 1976a, 1978, Seth 1977, Fujiwara & Kinoshita 1979]. Deterministic TC testing of sequential circuits has received little attention [Venkatraman & Saluja 1980]. In general it is known that for combinational circuits, TC testing can provide essentially the same fault coverage as conventional testing with a very small increase in the number of test patterns required. The main problem is the computational effort needed to compute the test patterns and the order in which they must be

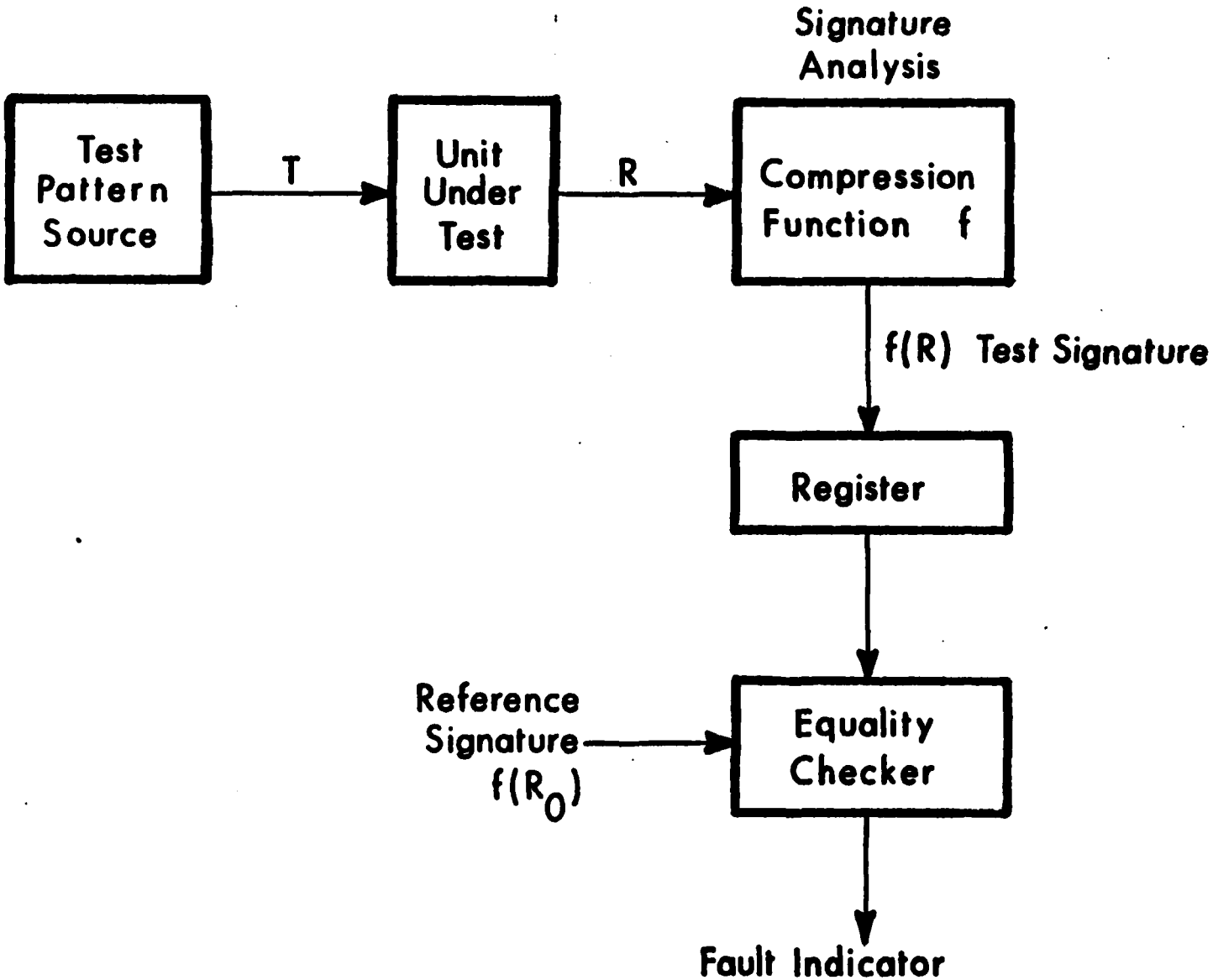


Fig. 3.4. Typical set-up for compact testing

applied. Note that even combinational TC tests are highly influenced by the order of the input patterns. A related testing technique called ones counting has also received some attention [Hayes 1976b, Losq 1978]. In ones counting f(R) is the number of 1's appearing in R. Hayes has analyzed ones counting from a deterministic viewpoint and concluded that it is potentially superior to TC testing [Hayes 1976b]. For example, in combinational circuits the ones count is independent of the order in which the input test patterns are applied. An advantage of transition counts is the fact that they are independent of 0 or 1 pulse duration, making them attractive for testing asynchronous circuits.

To simplify the task of computing T for compact testing, two approaches have been investigated: exhaustive testing and (pseudo-) random testing. In each case, the test pattern source of Fig. 3.4 need only be a simple counter. Exhaustive testing, as discussed in Chap. 2, is feasible for combinational circuits with a moderate number of inputs n, say $n \leqq 32$. It has been observed that for special types of unate circuits, all stuck-at faults can be detected by applying all input patterns and observing the output ones count [Tsidon et al. 1978]. In the case of fanout-free circuits it suffices to perform the counting modulo two, thus reducing the response recording circuitry to a single flip-flop.

Savir of IBM has defined a type of ones count $S = K/2^n$ for an n-input combinational circuit, where K is the number of minterms in the function being realized [Savir 1980]. S, which is called a syndrome, is thus the normalized number of ones observed at the circuit's output when all 2^n possible input combinations are applied. A circuit is syndrome-testable for a fault F, if F produces a syndrome S that differs from the fault-free syndrome S_0. S has the advantage of depending only on the function being realized and not its particular implementation. On the other hand, it is possible for faults to yield the same fault-free syndrome S_0 as the good circuit. Savir has shown that any combinational circuit can be made syndrome testable by the addition of a modest amount of extra logic to the unit under test. For example, the 74181, a standard 4-bit ALU chip, requires one extra input line and two extra gates to make it syndrome-testable with respect to stuck-at faults.

A comprehensive analysis of the use of random test generation with ones counting has been made by Losq [Losq 1978]. He considers the testing of sequential circuits, and assumes that testing is implemented in three steps as follows:

(1) (Initialization) A very long sequence of inputs is applied to bring the unit under test to a well-defined initial state. (This step is omitted in the case of combinational circuits.)

(2) (Statistics gathering) Another long sequence of inputs is applied and the ones count of the response sequence is computed.

(3) (Comparison) The computed ones count is compared to a reference signature to determine whether the unit under test is faulty.

Losq does not require an exact match between the observed and reference signatures; they may differ by some tolerance factor ε. The unit under test is considered to be fault-free only if its ones count falls within some acceptable range determined by ε.

To analyze the efficiency of this testing procedure a fault model that assigns occurrence probabilities to all faults under consideration is required. Losq considers several possible models of this type. For example, in the case of n-input combinational circuits he assumes that all possible fault functions have equal probability, namely $1/(2^{2^n}-1)$. For the more difficult task of modeling faults in sequential circuits, Losq considers separately the output circuits, the memory (flip-flop) part, and the excitation circuits. For each case he derives expressions for the probability that a fault is not detected. Figure 3.5 shows a representative result for a 1K-bit ROM [Losq 1978]. This indicates that when the length of the random input sequence to the ROM is 10^6, the probability of detecting all faults is 0.99999. Losq concludes that random compact testing can provide very high fault coverage for combinational circuits, while in the case of sequential circuits the coverage depends on the ease with which the unit under test can be initialized.

SIGNATURE ANALYSIS

The term signature analysis has been appropriated by Hewlett-Packard for the compact testing scheme depicted in Fig. 3.6 [Gordon & Nadig 1977, Hewlett-Packard 1977]. The test response R is passed through a 16-bit feedback shift

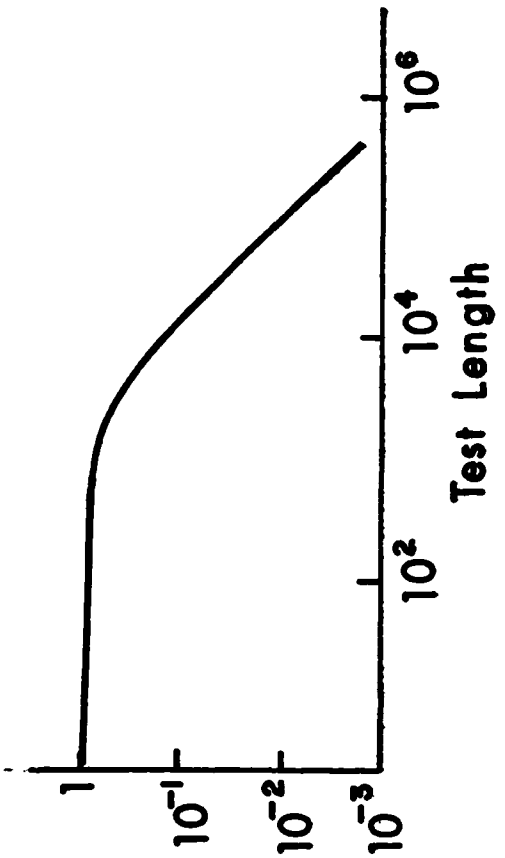


Fig. 3.5. Effectiveness of random compact testing for a 1K-bit ROM [Losq 1978].

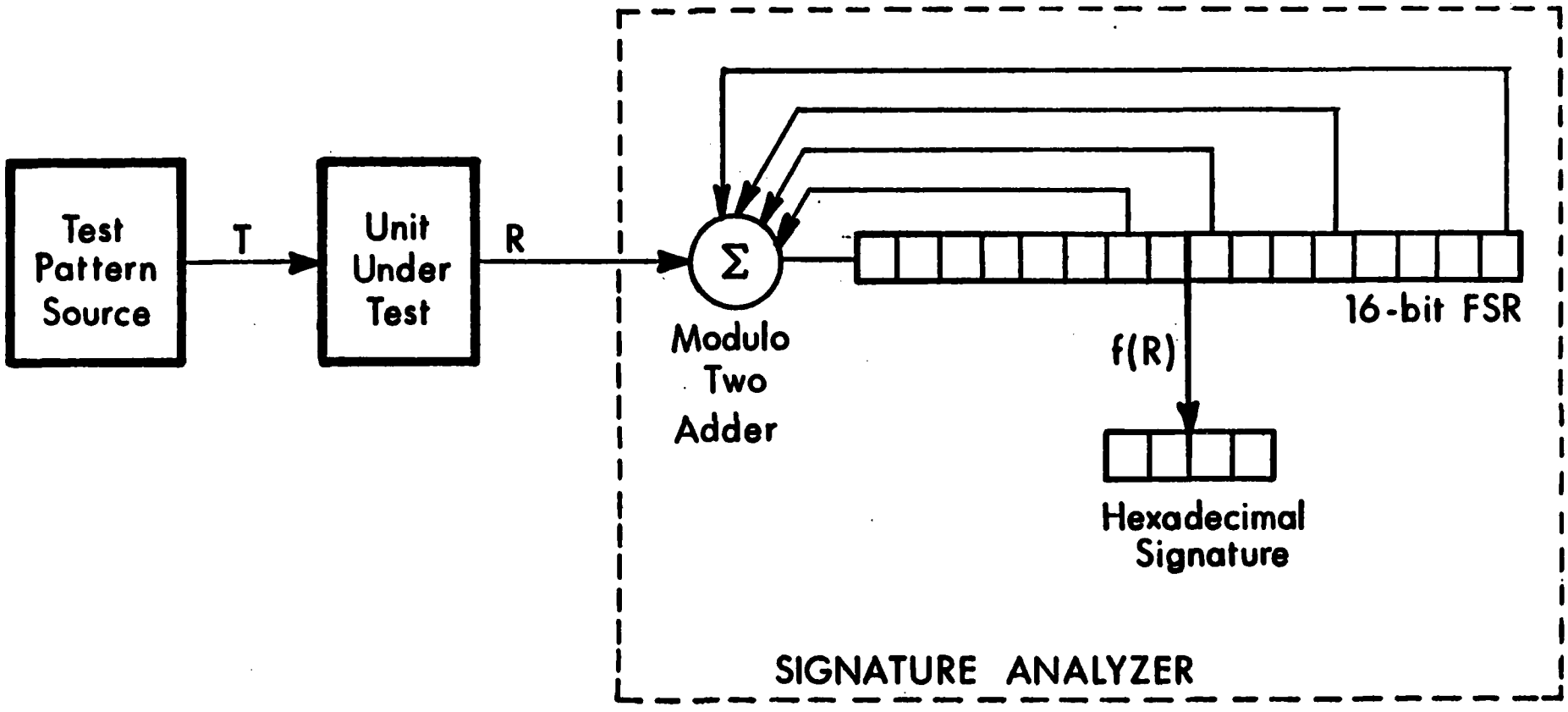


Fig. 3.6. Testing with a Hewlett-Packard signature analyzer.

register (FSR). The contents of the FSR after the complete test T has been applied is the fault signature f(R), which is then displayed as a 4-digit hexadecimal number. Signature analysis is based on algebraic coding techniques, since the FSR implements a decoding scheme employing polynomial division. The effectiveness of signature analysis can therefore be analyzed using standard methods from algebraic coding theory.

It is readily shown that the scheme of Fig. 3.6 can detect any single-bit error in the response stream R. If $|R| = k$, and all response streams are equally likely, then the probability of detecting an error with an r-bit signature is $1-(2^{k-r}-1)/(2^k-1)$, which approaches $1-2^{-r}$ as k, the test length, increases. Using this result, it has been argued that the HP signature analysis method with $r = 16$ has a probability of detecting faults of 99.998% [Gordon & Nadig 1977], and is far superior to transition counting. Several fallacies in this argument have been pointed out recently [Smith 1980]. In the first place, if all errors are equally likely, then signature analysis or long test sequences are not necessary. Equally good results could be obtained by using a test sequence of length r with no response compaction! While the assumption of independent errors is valid for the communication channels where algebraic coding methods are often employed, its validity is questionable in the case of digital logic circuits. Errors in test response streams from logic circuits often take the form of burst errors (several adjacent bits faulty) or "repeated-use" faults where errors occur in bit positions that are a fixed distance apart; typically this distance is a power of two. Smith has analyzed the ability of signature analysis to detect such faults. He concludes that results are highly dependent on the FSR configuration used in the signature analyzer, and that further experimental data is needed to identify the most efficient configurations.

Signature analysis appears to be a promising approach to the design of self-testing LSI/VLSI chips because of the relatively small amount of special hardware needed, particularly if pseudo-random test pattern generation is used. Recently some work has been reported on its application to the design of self-testing processors [Könemann et al. 1979]. The AMD 2901 4-bit microprocessor slice was redesigned to include a random pattern generator and a signature analyzer. Both units were implemented in the form of feedback shift registers obtained by modifying existing registers in the 2901. While encouraging results have been reported for this approach, its fault coverage is

questionable in view of the known limitations of random testing, and the limitations of signature analysis pointed out by Smith.

4. SPECIAL CIRCUITS AND FAULTS

The preceding chapters have dealt with test generation methods for general types of combinational and sequential circuits. The fault models considered included the general functional fault model of Sec. 3.1, and the stuck-at model which is the "classical" fault model of most test generation procedures. The present chapter examines test generation for a variety of special types of circuits including semiconductor memories and PLAs. It also examines the test generation requirements of such non-classical faults as delay faults and pattern sensitive faults which are of growing importance in the context of LSI/VLSI.

4.1 SEMICONDUCTOR MEMORIES

Considerable attention has been devoted to test generation for semiconductor memories including RAMs, ROMs, and PROMs [Hnatek 1975, Barraclough et al. 1976]. This reflects the enormous number of memory chips used in modern digital systems, as well as some special testing problems that they pose. These problems include pattern sensitivity, and soft errors due to α-particle radiation from chip packaging materials. Both of these problems are indirect consequences of the very small size of the memory cells used, and the very high density of these cells on a typical memory chip. Figure 4.1 lists some common failure modes in semiconductor memories. Faults due to α-radiation generate transient errors which are usually tackled by special packaging techniques or the inclusion of single-error correcting logic in the RAM. The remaining faults are basically permanent, and are detected by the application of appropriate test patterns.

A variety of heuristic test procedures have been developed for semiconductor memories [Hnatek 1975]. These are characterized by the use of test pattern generation and verification algorithms that are easily programmed on standard automatic test equipment. Testing time is typically proportional to n or n^2, where n is the bit storage capacity of the memory under test. A simple RAM test called CHECKERBOARD proceeds as follows. 0's and 1's are written into alternating cell locations in the memory. Then the contents of each cell are read out and verified. The process is repeated with 0's

- Opens and shorts
- Inaccessible cells
- Multiple cell accesses
- Pattern sensitivity
- Slow sense amplifier response
- Loss of data due to refresh failure
- Loss of data due to α-radiation

Fig. 4.1. Some typical failure modes in semiconductor memories.

and 1's interchanged, thus requiring a total of 4n tests, where each test involves one write or one read-and-verify operation. CHECKERBOARD checks the basic memory operation, as well as the refresh function of dynamic memories. Failures in the addressing mechanism are often checked by means of a class of test algorithms called GALPAT. In a typical GALPAT test 0 (1) is written into every cell of the memory under test. A 1 (0) is then written into some test cell C_i. Each cell $C_j \neq C_i$ is read in turn to determine whether its contents were disturbed by the write operation addressed to C. After reading all the C_j's, C_i is read again to ensure that it is still correct. This process is repeated with every cell in the memory acting as the test cell. The time required by this GALPAT procedure is $2n^2+8n$.

Memory testing algorithms of the foregoing kind have two serious limitations. The testing time required by the more sophisticated algorithms like GALPAT can be excessive. For example, suppose that the foregoing GALPAT test is applied to an n-bit RAM at the rate of one test every 100 ns. A 1K RAM ($n = 2^{10}$) can be tested in less than a second; however, a 1M RAM ($n = 2^{20}$) would require over six hours at the same testing rate. A second difficulty is the fact that the fault coverage of these methods is unclear since there is no underlying fault model. In fact GALPAT was developed experimentally (by Macrodata, Inc.) to detect faults in a specific memory chip from one manufacturer. Its validity as a universal test for RAMs is therefore open to question.

4.2 PROGRAMMABLE LOGIC ARRAYS

Programmable logic arrays (PLAs) form a very popular structure for implementing control logic. A PLA typically takes the form of two connected grids of conductors called the AND and OR arrays as shown in Fig. 4.2. The PLA is programmed by means of switches (diodes, transistors, fusible links, etc.) that connect the vertical input/output lines to the horizontal lines. The AND-OR arrays form a 2-level logic circuit capable of realizing any combinational function. The addition of I/O registers as depicted in Fig. 4.2 allows any sequential logic function to be realized.

Besides faults that can be modeled by the normal stuck-at model, PLAs are subject to the following special failure modes: shorts between vertical or horizontal grid lines, missing switches at crosspoints, and extra switches at crosspoints.

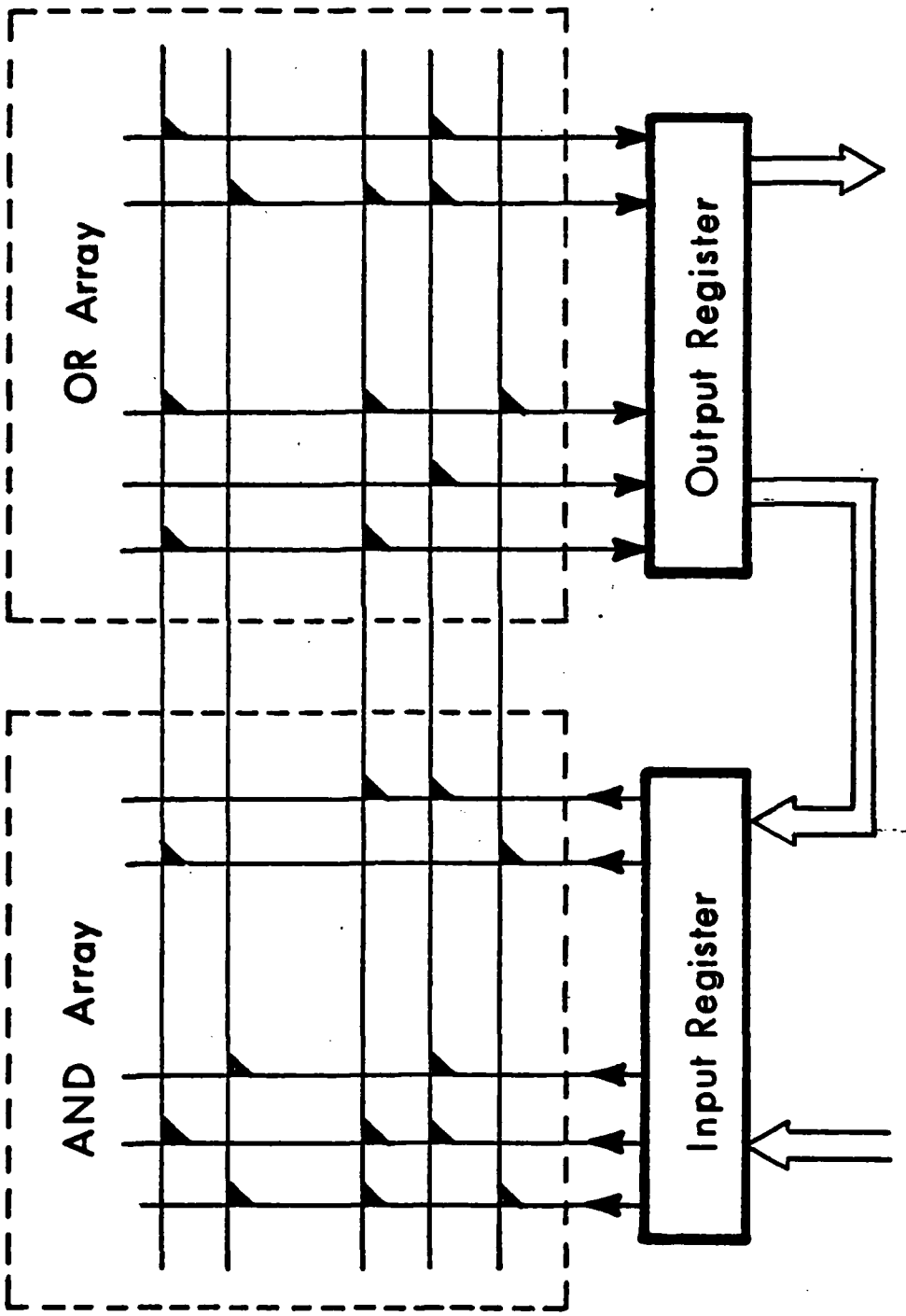


Fig. 4.2. Organization of a PLA.

Algorithms have been constructed for most PLA fault modes [Cha 1978, Ostapko & Hong 1979, Smith 1979]. These algorithms differ from conventional ones like the D-algorithm in that they do not employ an explicit logic circuit model of the PLA. Instead they use the knowledge of the chip layout to produce a set of test patterns for the most probable fault mechanisms. At the 1980 Fault-Tolerant Computing Symposium in Kyoto, two methods were proposed for designing PLAs to make test generation relatively trivial [Hong & Ostapko 1980, Fijiwara et al. 1980]. In each case a small amount of extra lgoic is added to a standard PLA layout. The augmented array has the property that a relatively small universal test set can be constructed for faults in the PLA such that the tests are independent of the functions being realized.

4.3 PATTERN-SENSITIVE FAULTS

An important failure mode encountered in semiconductor RAMs and ICs such as microprocessors that contain large internal RAMs, is the pattern sensitive fault (PSF) or adjacent pattern interference fault [Hnatek 1975, Hackmeister and Chiang 1975]. A PSF causes a memory read or write to fail in a way that depends on the stored pattern of information in the memory. PSFs are difficult to detect because a particular memory access operation say, read address X, can sometimes succeed and sometimes fail in the presence of a PSF. Thus although they are permanent faults, PSFs often have the appearance of intermittents. If the range of pattern sensitivity is broadly defined, then most of the failure modes listed in Fig. 4.1 can be covered by PSF tests.

A formal model and test generation procedure for PSFs was first proposed in 1975 [Hayes 1975]. A very general fault model was assumed in which each 1-bit cell C_i is associated with a set of cells $N(C_i)$ called the neighborhood of C_i. Any read or write operation addressed to C_i can alter or be altered by the state of $N(C_i)$ in any way. This is essentially a functional fault model in which PSFs are detected by applying a checking sequence to each neighborhood $N(C_i)$ of the RAM. Because of the regular structure of a typical RAM, such checking sequences are not difficult to generate; however their length increases exponentially with the neighborhood size. Thus in order to

obtain testing algorithms of practical length, constraints must be placed on neighborhood size or the number of possible fault modes permitted within a neighborhood.

Several memory testing procedures based on simplified PSF models have been studied. Nair et al. have defined PSFs based on a concept of cell coupling [Nair et al. 1978]. A set of k cells is said to be k-coupled if a state transition in one cell causes another cell in the set to change state. Test generation algorithms have been obtained to detect all 2-coupling faults and some 3-coupling faults. Nair et al. observe that their model becomes very complex when $k > 3$. Note that k-coupled faults exclude faults affecting read operations. A similar fault model has been investigated by Suk and Reddy, who also restrict attention to the neighborhood types appearing in Fig. 4.3 [Reddy & Suk 1979, Suk & Reddy 1980]. They provide efficient test generation algorithms for detecting and locating PSFs. However, the value of these methods is questionable because of the underlying assumption that reads and non-transition writes cannot change the state of the memory.

Recently Hayes has reconsidered his PSF model under the assumption that operations addressed to only one cell in the RAM can be faulty [Hayes 1980]. He showed that for a broad class of neighborhood shapes called tiling neighborhoods, e.g., those of Fig. 4.3, relatively simple and efficient test generation algorithms exist. In general, test sequence length is $(3k+2)2^k n$ where n is the RAM capacity and k is the neighborhood size. Thus for a 1M RAM and the 5-cell Type I neighborhood of Fig. 4.3, the total testing time is about 1 min at a 10 MHz testing rate. This compares very favorably with the heuristic RAM testing methods discussed earlier. The validity of the Hayes fault model, as well as those of other researchers has not been verified experimentally, so the practical value of the associated test procedures has not been determined.

4.4 DELAY FAULTS

The operating speed of a digital system is determined by the maximum propagation delays in the combinational parts of the system. Changes in propagation delay due to manufacturing defects or aging, which are called

Fig. 4.3. Simple RAM neighborhoods for PSF testing.

delay faults, can result in improper output signals or unstable operation. The goal of delay testing is to determine whether the delays of a given logic network lie within specifications. Delay testing is accomplished by measuring maximum propagation delays between the inputs and outputs of the subcircuits of interest. This may be difficult or impossible to do in LSI/VLSI chips if sufficient access to input/output lines within the chip is not available. The necessary access is provided by design disciplines such as LSSD, and delay fault testing for such circuits has been studied [Hsieh et al. 1977, Lesser & Shedletsky 1980].

A basic requirement for proper operation of logic circuits in that its maximum internal propagation delay d_{max} should not exceed one clock period, so that all flip-flop input signals are stable when the circuit is clocked. To keep the computation associated with delay measurement within practical limits, only signal delay along single (sensitized) paths is considered. The delay of such a path P from x to y is measured as follows. Using a procedure analogous to the D-algorithm, an input vector X required to sensitize P is determined. A second vector X′ is obtained by inverting the value of x in X. When the sequence XX′ is applied to the circuit under test, a signal transition propagates along P. The delay d between the application of XX′ and the appearance of a signal transition at y is then computed; this is the delay of P. If the delays of a sufficiently large and representative set of paths are determined, d_{max} can be computed. Note that it is not usually feasible to consider all possible paths.

A basic difficulty in delay testing is choosing the set of single paths whose delays are to be measured in the foregoing manner. Figure 4.4 illustrates this problem. The delays of the various gates are denoted by d and should not exceed 5; the delay of any input/output path should not exceed 20. There is a delay fault in G_4 where $d = 6$. Suppose that we attempt to delay-test this circuit by computing the delays of the two paths $G_1G_3G_5G_6$ and $G_2G_3G_4G_6$ which include every gate at least once. An acceptable delay of 20 is obtained in each case since the delay $d = 4$ in G_2 compensates for the delay fault in G_6. This circuit may operate improperly if the path $G_1G_3G_4G_6$ with delay 21 is sensitized during normal operation. Another problem encountered in delay testing is that certain paths cannot be sensitized by themselves

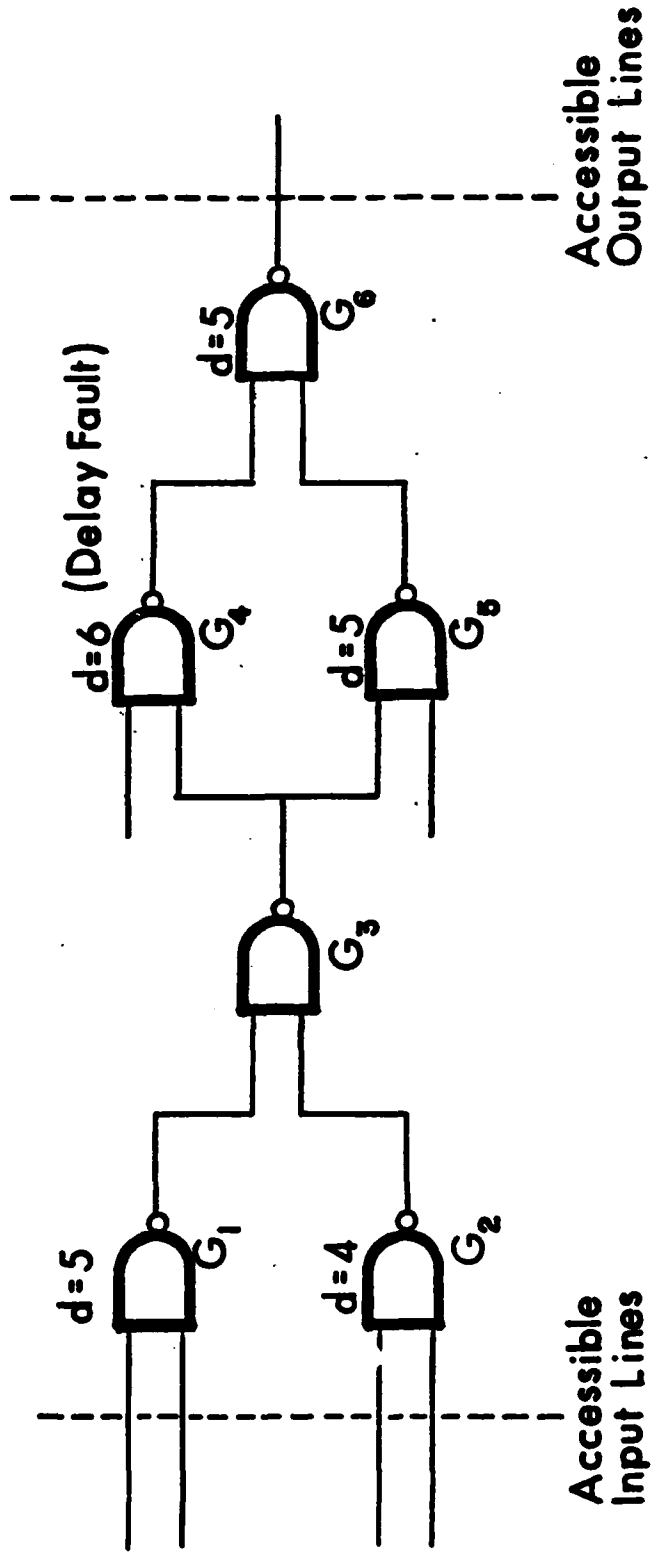


Fig. 4.4. Illustration of delay testing.

because of the multiple-path sensitization problem mentioned in Chap.2.

Hsieh et al. of IBM have devised a heuristic delay test generation method for LSSD-based systems [Hsieh et al. 1977]. They attempt to avoid the problem illustrated in Fig. 4.4 by including each gate G in two paths P_1 and P_2 which are as disjoint as possible. P_1 and P_2 are chosen to be the longest and shortest paths containing G whose delays are measurable via single-path sensitization.

A more systematic approach to delay testing has been developed recently by Shedletsky, also at IBM [Lesser & Shedletsky 1980]. Each path whose delay can be measured is decomposed into segments, each of which lies in a fanout-free subnetwork of the circuit under test. The delay of a path P which is not explicitly computed via path sensitization can be determined from the known delays of certain sets of paths $P_1, P_2, P, \ldots, P_k$ whose segments include all the segments of P. In general, the delay of P can be expressed as a linear combination of the delays of a suitable set of paths. To perform this calculation, a covering table for the measured paths and their segments is constructed; this is called a path matrix. Figure 4.5 shows a path matrix for the circuit of Fig. 4.4 where $d(P_i)$ denotes the delay of path P_i. The row $P_4 = G_1G_3G_4G_6$, which is missing from Fig. 4.5, can be expressed as a linear combination of the rows in Fig. 4.5 thus

$$P_4 = P_1 + P_2 - P_3 .$$

It is easily seen that the corresponding delays are related in the same fashion, viz.

$$\begin{aligned} d(P_4) &= d(P_1) + d(P_2) - d(P_3) \\ &= 20 + 20 - 19 \\ &= 21 . \end{aligned}$$

Note that $d(P_4)$ is identified as an unacceptable delay, even though P_4 is not explicitly sensitized. Furthermore, it is possible to compute the delays of paths that cannot be sensitized directly.

A delay test set is complete if the measured paths include a basis of the complete path matrix M for the circuit being delay tested. If R is the

Path	Segment				Path Delay
	G_1G_3	G_2G_3	G_4G_6	G_5G_6	
$P_1 = G_1G_3G_5G_6$	1	0	0	1	20
$P_2 = G_2G_3G_4G_6$	0	1	1	0	20
$P_3 = G_2G_3G_5G_6$	0	1	0	1	19

Figure 4.5. Path matrix for the circuit of Fig. 4.4.

rank of M, i.e., the size of any basis, then Shedletsky has shown that

$$R = 2(pi + \sum_{j}(fob_j - 1))$$

where pi is the number of primary inputs, fob_j is the fanout index of line j, and the summation is over all lines j that fan out. In some cases the number of single paths that can be sensitized is less than R, in which case the circuit in question is incompletely testable. Shedletsky's method has been implemented in a program DTEST. An experiment with six combinational circuits whose complexity ranged from 827 to 1248 gates showed that DTEST achieved an average delay fault coverage of 90.5 percent [Lesser & Shedletsky 1980].

5. BIBLIOGRAPHY

[Abramovici & Breuer 1980] M. Abramovici & M.A. Breuer: "Fault diagnosis based on effect-cause analysis: an introduction," *Proc. 17th Design Automation Conf.*, Minneapolis, pp. 69-76, June 1980.

[Agrawal 1978] V.D. Agrawal: "When to use random testing," *IEEE Trans. on Computers*, vol. C-27, pp. 1054-1055, November 1978.

[Agrawal & Agrawal 1972] V.D. Agrawal & P. Agrawal: "An automatic test generation system for ILLIAC IV logic boards," *IEEE Trans. on Computers*, vol. C-21, pp. 1015-1017, September 1972.

[Akers 1976] S.B. Akers: "A logic system for fault test generation," *IEEE Trans. on Computers*, vol. C-25, pp. 620-630, June 1976.

[Akers 1978a] S.B. Akers: "Functional testing with binary decision diagrams," *Journal Design Autom. & Fault-Tolerant Computing*, vol. 2, pp. 311-331, October 1978.

[Akers 1978b] S.B. Akers: "Binary decision diagrams," *IEEE Trans. on Computers*, vol. C-27, pp. 509-516, June 1978.

[Barraclough et al. 1976] W. Barraclough, A.C.L. Chiang & W. Sohl: "Techniques for testing the microcomputer family," *Proc. IEEE*, vol. 64, pp. 943-950, June 1976.

[Breuer 1971] M.A. Breuer: "A random and an algorithmic technique for fault detection test generation for sequential circuits," *IEEE Trans. on Computers*, vol. C-20, pp. 1364-1370, November 1971.

[Breuer & Friedman 1976] M.A. Breuer & A.D. Friedman: *Diagnosis and Reliable Design of Digital Systems*, Woodland Hills, CA, Computer Science Press, 1976.

[Breuer & Friedman 1979] M.A. Breuer & A.D. Friedman: "TEST/80 - a proposal for an advanced automatic test generation system," *Proc. IEEE Int'l. Automatic Testing Conf. (AUTOTESTCON '79)*, Minneapolis, September 1979.

[Breuer & Friedman 1980] M.A. Breuer & A.D. Friedman: "Functional level primitives in test generation," *IEEE Trans. on Computers*, vol. C-29, pp. 223-235, March 1980.

[Breuer & Harrison 1974] M.A. Breuer & L. Harrison: "Procedures for eliminating static and dynamic hazards in test generation," *IEEE Trans. on Computers*, vol. C-23, pp. 1069-1078, October 1974.

[Cha 1978] C.W. Cha: "A testing strategy for PLAs," *Proc. 15th Design Automation Conf.*, Las Vegas, pp. 326-334, June 1978.

[Cha et al. 1978] C.W. Cha, W.E. Donath & F. Özgüner: "9-V algorithm for test pattern generation of combinational digital circuits," *IEEE Trans. on Computers*, vol. C-27, pp. 193-200, March 1978.

[Chappell et al. 1977] S.G. Chappell et al.: "Functional simulation in the LAMP system," *Journal of Design Automation & Fault-Tolerant Computing*, vol. 1, pp. 203-215, May 1977.

[El-Ziq 1979] Y.M. El-Ziq: "Testing of MOS combinational networks: a procedure for efficient fault simulation and test generation," *Proc. 16th Design Automation Conf.*, San Diego, pp. 162-170, June 1979.

[Friedman & Menon 1971] A.D. Friedman & P.R. Menon: *Fault Detection in Digital Circuits*, Englewood Cliffs, NJ, Prentice-Hall, 1971.

[Fujiwara & Kinoshita 1979] H. Fujiwara & K. Kinoshita: "Testing logic circuits with compressed data," *Journal of Design Automation & Fault-Tolerant Computing*, vol. 3, pp. 211-225, Winter 1979.

[Fujiwara et al. 1980] H. Fujiwara, K. Kinoshita & H. Ozaki: "Universal tests for programmable logic arrays," *Digest 10th Fault-Tolerant Computing Symp.*, Kyoto, pp. 137-142, October 1980.

[Galiay et al. 1980] J. Galiay, Y. Crouzet & M. Vergniault: "Physical versus logical fault models in MOS LSI circuits: impact on their testability," *IEEE Trans. on Computers*, vol. C-29, pp. 527-531, June 1980.

[Goel 1980] P. Goel: "An implicit enumeration algorithm to generate tests for combinational logic circuits," *Digest 10th Fault-Tolerant Computing Symp.*, Kyoto, pp. 145-150, October 1980.

[Gordon & Nadig 1977] G. Gordon & H. Nadig: "Hexadecimal signatures identify troublespots in microprocessor systems," *Electronics*, vol.50, no.5, pp. 89-96, March 3, 1977.

[Hackmeister & Chiang 1975] D. Hackmeister & A.C.L. Chiang: "Microprocessor test technique reveals instruction pattern sensitivity," *Computer Design*, vol. 14, no. 12, pp. 81-85, December 1975.

[Hayes 1971] J.P. Hayes: "On realizations of Boolean functions requiring a minimal or near minimal number of tests," *IEEE Trans. on Computers*, vol. C-20, pp. 1506-1513, December 1971.

[Hayes 1975] J.P. Hayes: "Detection of pattern sensitive faults in random access memories," *IEEE Trans. on Computers*, vol. C-24, pp. 150-157, February 1975.

[Hayes 1976a] J.P. Hayes: "Transition count testing of combinational logic circuits," *IEEE Trans. on Computers*, vol. C-25, pp. 613-620, June 1976.

[Hayes 1976b] J.P. Hayes: "Check sum methods for test data compression," *Journal Design Automation & Fault-Tolerant Computing*, vol. 1, pp. 3-17, October 1976.

[Hayes 1978] J.P. Hayes: "Generation of optimal transition count tests," *IEEE Trans. on Computers*, vol. C-27, pp. 36-41, January 1978.

[Hayes 1980] J.P. Hayes: "Testing memories for single-cell pattern-sensitive faults," *IEEE Trans. on Computers,* vol. C-29, pp. 249-254, March 1980.

[Hayes & McCluskey 1980] J.P. Hayes & E.J. McCluskey: "Testability considerations in microprocessor-based design," *Computer,* vol. 3, no. 3, pp. 17-26, March 1980.

[Hewlett-Packard Co. 1977] Hewlett-Packard Company: "A Designer's Guide to Signature Analysis, Application Note 222, Palo Alto, CA, April 1977.

[Hill & Huey 1977] F.J. Hill & B. Huey: "SCIRTSS: a search system for sequential circuit test sequences," *IEEE Trans. on Computers,* vol. C-26, pp. 490-502, May 1977.

[Hnatek 1975] E.R. Hnatek: "4-kilobit memories present a challenge to testing," *Computer Design,* vol. 14, pp. 117-125, May 1975.

[Hong & Ostapko 1980] S.J. Hong & D.L. Ostapko: "FITPLA: a programmable logic array for function independent testing," *Digest 10th Fault-Tolerant Computing Symp.,* Kyoto, pp. 131-136, October 1980.

[Hsieh et al. 1977] E.P. Hsieh et al.: "Delay test generation," *Proc. 14th Design Automation Conf.,* New Orleans, pp. 486-491, June 1977.

[Huey 1979] B.M. Huey: "Heuristic weighting functions for guiding test generation searches," *Journal Design Automation & Fault-Tolerant Computing,* vol. 3, pp. 21-39, January 1979.

[Könemann et al. 1979] B. Könemann, J. Mucha & G. Zweihoff: "Built-in logic block observation techniques," *Proc. 1979 Test Conf.,* Cherry Hill, NJ, pp. 37-41, October 1979.

[Lesser & Shedletsky 1980] J.D. Lesser & J.J. Shedletsky: "An experimental delay test generator for LSI logic," *IEEE Trans. on Computers,* vol. C-29, pp. 235-248, March 1980.

[Losq 1978] J. Losq: "Efficiency of random compact testing," *IEEE Trans. on Computers,* vol. C-27, pp. 516-525, June 1978.

[Lyons 1974] N.P. Lyons: "FAULTRACK: universal fault isolation procedure for digital logic," *1974 IEEE Intercon Tech. Program,* New York, paper no. 40/2 [9p], March 1974.

[McCaskill 1976] R. McCaskill: "Test approaches for four-bit microprocessor slices," *1976 Semiconductor Test Symposium Digest,* pp. 22-24, October 1976.

[Mead & Conway 1980] C. Mead & L. Conway: *Introduction to VLSI Systems,* Reading, MA, Addison-Wesley, 1980.

[Mick & Brick 1980] J. Mick & J. Brick: *Bit-Slice Microprocessor Design*, New York, McGraw-Hill, 1980.

[Muth 1976] P. Muth: "A nine-valued circuit model for test generation," *IEEE Trans. on Computers*, vol. C-25, pp. 630-636, June 1976.

[Nair et al. 1978] R. Nair, S.M. Thatte & J.A. Abraham: "Efficient algorithms for testing semiconductor random-access memory," *IEEE Trans. on Computers*, vol. C-27, pp. 572-576, June 1978.

[Ostapko & Hong 1979] D.L. Ostapko & S.J. Hong: "Fault analysis and test generation for programmable logic arrays (PLA)," *IEEE Trans. on Computers*, vol. C-28, pp. 617-627, September 1979.

[Parker 1976a] K.P. Parker: "Compact testing: testing with compressed data," *Proc. 1976 Int'l. Symp. on Fault-Tolerant Computing*, Pittsburgh, pp. 93-98, June 1976.

[Parker 1976b] K.P. Parker: "Adaptive random test generation," *Journal Design Automation & Fault-Tolerant Computing*, vol. 1, pp. 62-83, October 1976.

[Putzolu & Roth 1971] G.F. Putzolu & J.P. Roth: "A heuristic algorithm for the testing of asynchronous circuits," *IEEE Trans. on Computers*, vol. C-20, pp. 639-647, June 1971.

[Reddy & Suk 1979] S.M. Reddy & D.S. Suk: "Test procedures for semiconductor random access memories," Rome Air Development Center, Technical Report RADC-TR-79-269, November 1979.

[Roth 1966] J.P. Roth: "Diagnosis of automata failures: a calculus and a method," *IBM Journal Res. & Dev.*, vol. 10, pp. 278-291, 1966.

[Roth 1980] J.P. Roth: *Computer Logic, Testing and Verification*, Potomac, MD, Computer Science Press, 1980.

[Savir 1980] J. Savir: "Syndrome testable design of combinational circuits," *IEEE Trans. on Computers*, vol. C-29, pp. 442-451, June 1980.

[Schnurmann et al. 1975] H.D. Schnurmann, E. Lindbloom & G.G. Capenter: "The weighted random test pattern generator," *IEEE Trans. on Computers*, vol. C-24, pp. 695-700, July 1975.

[Seth 1977] S.C. Seth: "Data compression techniques in logic testing: an extension of transition counts," *Journal of Design Automation & Fault-Tolerant Computing*, vol. 1,pp. 99-114, February 1977.

[Smith 1979] J.E. Smith: "Detection of faults in programmable logic arrays," *IEEE Trans. on Computers*, vol. C-28, pp. 845-853, November 1979.

[Smith 1980] J.E. Smith: "Measures of effectiveness of fault signature analysis," *IEEE Trans. on Computers*, vol. C-29, pp. 510-514, June 1980.

[Snethen 1977] T.J. Snethen: "Simulation-oriented fault test generator," *Proc. 14th Design Automation Conf.*, New Orleans, pp. 88-93, June 1977.

[Sridhar & Hayes 1979] T. Sridhar & J.P. Hayes: "Testing bit-sliced microprocessors," *Digest 9th Int'l. Symp. on Fault-Tolerant Computing*, Madison, WI, pp. 211-218, June 1979.

[Sridhar & Hayes 1981] T. Sridhar & J.P. Hayes: "A functional approach to testing bit-sliced microprocessors," *IEEE Trans. on Computers*, 1981, to appear.

[Srini 1977] V.P. Srini: "API tests for RAM chips," *Computer*, vol. 10, no. 7, pp. 32-35, July 1977.

[Suk & Reddy 1980] D.S. Suk & S.M. Reddy: "Test procedures for a class of pattern sensitive faults in semiconductor random access memories," *IEEE Trans. on Computers*, vol. C-29, pp. 419-429, June 1980.

[Thatte 1979] S.M. Thatte: "Test Generation for Microprocessors," Ph.D. Thesis, University of Illinois, Department of Electrical Engineering, 1979; also, Coord. Science Lab. Report R-842, May 1979.

[Thatte & Abraham 1980] S.M. Thatte & J.A. Abraham: "Test generation for microprocessors," *IEEE Trans. on Computers*, vol. C-29, pp. 429-441, June 1980.

[Thomas 1971] J.J. Thomas: "Automated diagnostic test programs for digital networks," *Computer Design*, pp. 63-67, August 1971.

[Tsidon et al. 1978] A. Tsidon, I. Berger & M. Yoeli: "A practical approach to fault detection in combinational networks," *IEEE Trans. on Computers*, vol. C-27, pp. 968-971, October 1978.

[Vaughn 1976] G.D. Vaughn: "CDALGO - a test pattern generation program," *Proc. 13th Design Automation Conf.*, San Francisco, pp. 186-193, June 1976.

[Wadsack 1978] R.L. Wadsack: "Fault modeling and logic simulation of CMOS and MOS integrated circuits," *BSTJ*, vol, 57, pp. 1449-1474, May/June 1978.

[Williams & Eichelberger 1977] T.W. Williams & E.B. Eichelberger: "Random test patterns within a structured sequential logic network," *Digest Semiconductor Test Symp.*, Cherry Hill, NJ, pp. 19-27, October 1977.

[Yamada et al. 1977] A. Yamada et al.: "Automatic test generation for large digital circuits," *Proc. 14th Design Automation Conf.*, New Orleans, pp. 79-83, June 1977.

END

FILMED

6-83

DTIC